НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації і управління

УДК: 004

«До захисту допущено»
В.о. завідувача кафедри
_____ *О.А.Павлов*
(підпис) (ініціали, прізвище)
“___” _____2019 р.

Дипломний проект
на здобуття ступеня бакалавра

з напряму підготовки *6.050101 «Комп’ютерні науки»*

на тему:«*Автоматизоване робоче місце музичного редактора*»

Виконав: студент *4* курсу, групи *ІС-з51в*
Ступинець Наталія Василівна _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *ст. викладач Халус О.А.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації *доц., к.т.н., доц. Тєлишева Т.О.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент *доц. каф. ТК, к.т.н. Тимошин Ю.А.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент Ступинець Н.В. _____
(підпис)

Київ – 2019 р.

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проекту складається з чотирьох розділів, містить 24 рисунки, 17 таблиць, 1 додаток, 9 джерел.

Дипломний проект присвячений розробці програмного забезпечення для автоматизації роботи музичного редактора на радіостанції.

У розділі інформаційного забезпечення були визначені вхідні та вихідні дані для реалізації всіх необхідних функцій, була розроблена структура xml-файлів для збереження даних.

У розділі «Програмне та технічне забезпечення» були описані небхідні технічні характеристики ПК, для можливості функціонування застосунку, що автоматизуватиме роботу музичного редактора, а також основні засоби розробки даного застосунку.

У технологічному розділі була описана перевірка функціональності застосунку, інструкція з його використання.

ЗАВАНТАЖЕННЯ ФАЙЛІВ, URL, ПЛЕЙЛИСТ, ЛОКАЛЬНА МЕРЕЖА, БАГАТОПОТОЧНІСТЬ, QT ФОРМИ, КЛІЄНТ-СЕРВЕР, ВІНДОВС

.

					ДП ІС-з4119.1722-с.ПЗ		
		Прізвище	Підпис	Дата			
Розроб.		*Ступинець Н.В.*			Автоматизоване робоче місце музичного редактора	*Літ.*	*Арк.* 2 / *Аркушів*
Перевірив.		*Халус О.А.*					
						КПІ ім. І.Сікорського ФІОТ кафедра АСОІУ гр. ІС-з41	
Н. кон.		*Халус О.А.*					
Затв.		*Павлов О.А.*					

ABSTRACT

The structure and the amount of work. The explanatory note of the diploma project consists of four chapters, contains 24 pictures, 17 tables, 1 addition, 9 sources.

The main goal of this project is to develop software for automation of radio station music editor's work.

Input and output data, which is needed for implementation of all necessary functions was defined in the 'Information support' chapter. The structure of xml-files for data storage was also described in this chapter.

All required technical characteristics of the PC, were described in the 'Software' chapter. These technical characteristics must be supported by the PC to let the application run on this PC and manage all its functions. Development tools were also described in scope of this chapter.

The check of application functionality and user instruction were described in the technical chapter.

FILES DOWNLOADING, URL, PLAYLIST, LOCAL NETWORK, MULTITHREADING, QT FORMS, CLIENT-SERVER, WINDOWS

ЗМІСТ

ВСТУП

Дипломний проект присвячений розробці автоматизованого робочого місця музичного редактора на радіостанції.

Цілі створення комплексу задач полягають у спрощенні виконання повсякденних функцій редактора та доведення їх до повної автоматизації, для того, щоб мінімізувати ризик винекнення помилок, спричинених людським фактором за рахунок того, що редактор постійно виконує одну і ту ж робоу і може втрачати увагу, виконувати дії неправильно.

Для досягнення цілей у роботі вирішуються наступні задачі: автоматичний пошук та скачування нової музики з інтегрованих ресурсів, для подальшого їх додавання до музичної бази радіостанції, автоматичне генерування плейлистів випадковим чином, але за умови дотримання певних правил складання плей-листів, визначених у шаблонах плей-листів, автоматичне інформування інших працівників радіостанції (радіоведучих, арт-директора, програмного директора) про оновлення в музичному архіві, шляхом надсилання їм листів на робочу пошту, створення додаткового процесу для безперервного радіомовлення на іншому комп’ютері у локальній мережі та можливість впливати на цей процес засобами застосунку музичного редактора.

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Музичний редактор на радіо виконує широкий спект діяльності, це людина, яка несе на собі велику відповідальність, але її робота наразі є практично не автоматизованою, більшість функцій доводиться виконувати мануально. Тому часто можуть виникати неприємні ситуації, помилки, пов’язані з цим людським фактором. Хоча, цього можна уникнути, адже робота, виконувана цією людиною, підлягає якщо не повній, то хоча б частковій автоматизації. Ця автоматизація могла би значно полегшити життя працівнику та зменшити кількість помилок, які він міг би допустити.

1.1.1 Опис процесу діяльності

- Пошук музики

Якщо не кожного дня, то хоча б декілька разів на тиждень, музичний редактор моніторить різноманітні музичні ресурси, особливо якщо це молодіжна радіостанція. Адже будь-якого дня може вийти новий сингл, а вам потрібно, щоб першим він прозвучав саме на вашій станції. Тож тут задача для автоматизації зрозуміла — кожного дня, години (залежить від потреб користувача) перевіряти певні ресурси на появу нових пісень, якщо вони є - завантажувати їх та пропонувати музичному редактору прослухати та підтвердити їх додавання до музичної бази (тут мається на увазі не база даних, а директорія з усіма піснями, які є в ротації, на робочій станції редактора), якщо він не підтвердить — видаляти їх. Також на етапі додавання нової пісні слід визначати певні теги для неї. Наприклад, чи пісня сумна, весела, підходить на траурний ефір чи виключно для святкового настрою вранці. Ця інформація буде корисною для нас під час наступного кроку («Складання плейлиста») . На додачу звіт про додані до музичної бази

пісні надсилати усій команді радіостанції на робочу пошту (до такої команди може входити радіоведучий(і), програмний директор, тощо).

- Складання плейлиста

Якщо музичний редактор щось переплутав та зробив плейлисти на два дні підряд ідентичними (що абсолютно неприпустимо по радійних мірках), то йому це можна пробачити, хоча така помилка не пройде непоміченою повз слухачів. Саме тому краще довірити цю роботу комп'ютеру, щоб уникнути таких ситуацій.

Тож, одна з основних функцій — генерування плейлиста за заданим шаблоном. В попередньому пункті були описані «теги», які будуть присвоюватися пісні на етапі її додавання, саме вони і знадобляться для створення шаблону. Шаблон плейлиста представлятиме собою набір таких тегів. При генерації плейлиста користувач зможе обрати теги які його задовільняють та теги, які потрібно виключити з пошуку. Всі інші теги ігноруватимуться. На рисунку 1.1 наведено приклади таких тегів.

Рисунок 1.1 – Приклади тегів для композицій

Окрім цього плейлист повинен генеруватися таким чином, щоб виконувати певні правила — не додавати одного виконавця двічі на годину, не додавати одну і ту саму пісню більше ніж раз на декілька годин, тощо.

Також при генеруванні плейлиста потрібно врахувати його тривалість. Тривалість згенерованого плейлиста повинна бути максимально наближеною до заданої користувачем.

Якщо автоматизувати цей процес, то людині, що займає цю посаду, вдасться зекономити декілька годин, адже все, що потрібно буде зробити — згенерувати плейлист випадковим чином, дотримуючись при цьому певних правил.

-Інформування інших членів команди про оновлення бази пісень

Часто після проробленої роботи, додавання нової музики, музичний редактор може забути поділитися з командою тими піснями, які були додані до бази радіостанції. Це погано, адже радіоведучі знаючи, що нового додалося, які пісні тепер можна буде почути на їх радіостанції, можуть розповісти про це слухачам та зацікавити їх таким чином. А програмний директор міг би внести корективи у цей список. Тому після оновлення бази потрібно надсилати результати всій команді на робочу пошту.

-Моніторинг директорій музичного архіву

Після того, як нові пісні були знайдені, вони додаються в музичний архів, який представляє собою директорію чи набір директорій на персональному комп’ютері музичного редактора. Якщо композиція була додана за допомогою нашого застосунку, то ми знатимемо все про цю композицію. Але що, якщо користувач додав до музичкої директорії пісню просто перемістивши її туди? Тоді ми нічого не знатимемо про неї і відповідно не зможемо використати її під час генерації плейлиста.

Відповідно, нам потрібно моніторити ці директорії на предмет додавання туди нових файлів. Якщо файли були додані – просити користувача заповнити інформацію про них.

-Надсилання музики та плейлистів через локальну мережу

Музичний редактор та ведучі не можуть одночасно працювати з одним ПК. Відповідно, на комп'ютері ведучих буде запущено інший процес, який відповідатиме за відтворення музики та проситиме дані про плейлисти та самі композиції у «сервера», роль якого відіграватиме основна аплікація.

Це також означає, що за нормальних умов аплікація повинна бути запущена постійно - для того, щоб постачати інформацію клієнту.

-Відтворення музики на клієнтській стороні

Клієнт просить у сервера інформацію про плейлист та пісні, які входять до цього плейлиста, а також відтворює ці пісні. Врато зазначити, що ці дії повинні відбуватися паралельно. Тобто, пісні повинні звучати завжди, адже в радійних ефірах пауз практично не буває. А паралельно з цим клієнт в певні моменти часу повинен просити у сервера нові дані та нові пісні.

-Запис та зчитування даних про музичний архів

Як вже було зазначено вище, за нормальних умов аплікація повинна бути запущеною завжди, адже в музичних ефірах немає пауз. Але бувають випадки, коли необхідно перезавантажити комп'ютер, на якому запущена програма, буває, що пропадає світло на довгий період часу, тощо. Тому потрібно застрахуватися від таких випадків та тримати таку інформацію в файлах, які зберігатимуться вже безпосередньо на жосткому диску. В цих файлах зберігатиметься інформація про всі пісні, які є в музичному архіві, плейлисти та дати на які вони заплановані, конфігурація тих чи інших модулів програми. Тому ще однією функцією застосунку буде зчитування

цих файлів та ініціалізація інших компонент згідно з конфігурацією зазначеною у них.

Наведемо структурну схемо діяльності на рисунку 1.2

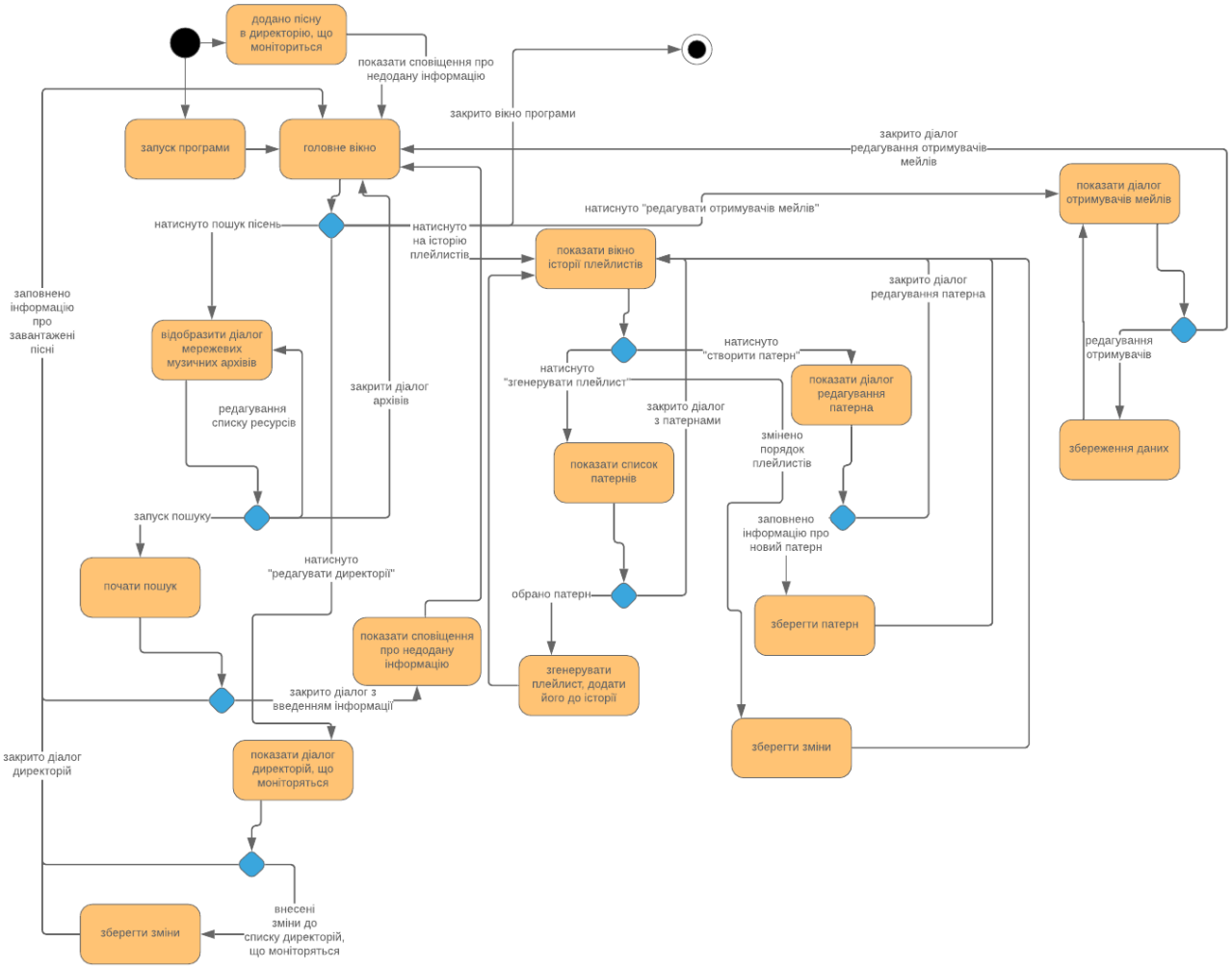


Рисунок 1.2 – Структурна схема діяльності

1.1.2 Опис функціональної моделі

Фактично, у нас буде одна система, яка складатиметься з двох аплікацій. Основний застосунок - робоче місце музичного редактора, він же в нашому випадку «сервер» та помічний, так званий клієнт, який проситиме інформацію у основного застосунку та відповідатиме за відтворення музики.

Актором основної системи є музичний редактор. Визначимо, які функції він виконує в цій системі, для цього наведемо таблицю 1.1, в якій описані актори, функції та їх описи.

Таблиця 1.1 – Опис функцій основної системи

Актор	Функція	Опис
Музичний редактор	Скачування нової музики з заданих ресурсів	Музичному редактору доступна опція ‘Start music search’, йому показується список ресурсів, з яких скачуватиметься нова музика. Він натискає кнопку ‘Start’ і найновіша музика скачується на ПК редактора.
	Вибір ресурсів	Музичному редактору доступна опція зміни ресурсів, з яких проводитиметься завантаження, ці дані зберігаються навіть після перезапуску застосунку.
	Надсилання листів з оновленнями	Музичний редактор може надіслати інформацію про оновлення в базі пісень іншим працівникам (радіоведучим, програмному директору, тощо).
	Генерування плейлистів	Музичний редактор може згенерувати плейлисти на майбутнє, використовуючи різноманітні кастомні шаблони плейлистів та конфігурацію.
	Створення шаблонів плейлистів	Музичний редактор має змогу створювати свої шаблони плейлистів, за якими згодом генеруватимуться плейлисти.

Продовження таблиці 1.1

Музичний редактор	Додавання інформації про нові пісні, що були додані до музичного архіву	Це стосується як пісень, що були скачані та додані за допомогою нашого застосунку, так і пісень, які були додані до музичного архіву шляхом переміщення музичного файлу в одну з папок, що моніторяться застосунком.
	Керування таймінгом плейлистів	Музичний редактор вирішуватиме коли та який плейлист прозвучить в ефірі шляхом додавання плейлистів до таймінгу.
	Надсилання плейлиста та пісень, що до нього входять, клієнту (ПК радіоведучого) у відповідності із заданим таймінгом для плейлистів	Всі пісні та інформація про плейлисти зберігатимуться на робочому комп’ютері редактора. Для того, щоб вони прозвучали в ефірі, вони мають бути надіслані клієнту (робочий комп’ютер радіоведучого).

Схема структурна варіантів використання наведена у графічному матеріалі [1].

Що стосується допоміжної системи (клієнтської сторони), то в цій системі єдина функція, яку небхідно буде виконати радіоведучому – запустити застосунок. Адже програвання пісень та запити на отримання інформації про нові плейлисти та власне пісні, які до нього входять, відбуватимуться без його участі.

1.2 Огляд наявних аналогів

Серед більшості українських радіостанцій («Люкс ФМ», «Хіт ФМ», «Фреш ФМ», «Стрий ФМ», «ФМ Галичина») найбільш широко використовується застосунок для автоматизації радіотрансляції - SoundEmpire[1]. У нього є окремий модуль для часткової автоматизації роботи музичного редактора, проте він не надто добре організований. Інтерфейс застосунку не є привабливим, а автоматизація є лише частковою.

Нижче наведено скріншот програми.

Рисунок 1.3 – Користувацький інтерфейс програми SoundEmpire (модуль для автоматизації роботи музичного редактора)

Нижче, у таблиці 1.2 наведемо порівняння функціональностей, які доступні у нашому застосунку та у застосунку SoundEmpire.

Таблиця 1.2 – Порівняння застосунку для автоматизації робочого місця музичного редактора та застосунку Sound Empire:

Функціональність	Наш застосунок	SoundEmpire
Автоматичне генерування плей-листів	доступне	доступне
Автоматичний пошук нових пісень в інтернеті	доступне	не досупне
Автоматичне сповіщення колег про зміни у музичній базі	доступне	не досупне
Радіотрансляція музики на комп'ютері клієнта (радіоведучого) з можливістю впливати на процес відтворення та коригувати його поведінку	досупне	досупне

Тож, наш застосунок можна класифікувати як унікальний, адже схожого до його функціоналу не має жоден застосунок, зокрема найпопулярніший у країнах СНД – Sound Empire. Що стосується українського ринку, то схожих застосунків у цьому сегменті взагалі не було знайдено.

1.3 Постановка задачі

1.3.1 Призначення розробки

Призначенням застосунку є автоматизація робочого місця музичного редактора на радіостанції.

1.3.2 Цілі та задачі розробки

Цілями розробки є:

- полегшення процесу пошуку нової музики для музичного редактора;
- полегшення процесу створення плей-листів музичним редактором;
- покращення процесу інформування інших учасників команди про останні зміни у музичному архіві;
- покращення користувацького досвіду музичного редактора шляхом створення зрозумілого та простого інтерфейсу;
- забезпечення безперервної радіотрансляції ефіру.

Для досягнення поставлених цілей необхідно розв’язати наступні задачі:

- для пошуку музики:

  а) пошук ресурсів з відкритим API для скачування та отримання додаткової інформації(про жанр, час додавання, тощо);

  б) реалізація цих процесів з використанням механізмів багатопоточності;

  в) вирішення проблеми повторного додавання до бази однієї і тієї ж пісні, а також проблеми її повторного скачування (якщо вона була видалена чи прийнята музичним редактором);

- для генерації плей-листів:

  а) уніфікувати створення шаблонів;

б) уніфікація додаткових правил для їх створення (щоб одна пісня не повторювалася двічі протягом декількох годин, щоб один виконавець не звучав в ефірі двічі на годину, тощо);

- для надсилання оновлених даних про базу пісень іншим учасникам команди:

а) імплементація методу розсилання листів за допомогою API for Google Gmail;

б) відтворення музичного контенту на іншому ПК в локальній мережі з можливістю музичного редактора впливати на цей процес.

Висновок до розділу

В цьому розділі було описано причину вибору саме такого об’єкту автоматизації, функції реального працівника, які може частково або повнісю виконувати програма, предметне середовище.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вхідні дані

Як вже зазначалося вище, за нормальних умов застосунок повинен постійно бути запущеним, адже радіоефіри не мають перерв у трансляції. Але бувають різні ситуації, такі як: припинення подачі електроенергії на тривалий час, необхідність перезавантажити ПК, тощо. За таких умов потрібно щоб основна конфігурація та інформація про систему зберігалася у постійній пам’яті – на жосткому диску. І при запуску застосунку ми би зчитували цю інформацію.

Одним з найбільш зручних сопособів зберігання такої інформації є XML-файли. Саме в них і будемо зберігати вхідні дані для нашого застосунку. Перелік потреб для яких ми використовуватимемо файли з вхідними даними наведені нище:

1) **Інформація про всі пісні, що належать до музичної бази**

   Для того, щоб після запуску застосунку заново не проходитися по всіх папках, що належать до музичного архіву, ми зберігатимемо інформацію про всі пісні у файлі. Також окрім назви пісні та її тривалості, які ми могли б дізнатися і без допомоги додаткових файлів, для нас ще важливі її теги. Саме згідно з цими тегами генеруватимуться плейлисти. Тож для того, щоб після запуску користувач не вводив заново інформацію про пісні, стає очевидно, що без файлу тут не обійтися. Назвемо його «songManager.xml». Структура файлу та візуальна схема будуть наведені нижче, у підпункті «Структура масивів інформації».

2) **Інформація про працівників, що отримуватимуть сповіщення на робочу пошту**

Імена та адреси поштових скриньок працівників, що будуть інформуватися про зміни у музичному архіві, також зберігатимемо у файлі. Для того, щоб кожного разу після перезапуску застосунку не вводити всю інформацію ще раз, а робити це лише за потреби – коли потрібно додати або видалити співробітника. Дані зберігатимуться в окремому XML-файлі, назвемо його «emailNotifierData.xml». Структура файлу та візуальна схема будуть наведені нижче, у підпункті «Структура масивів інформації».

3) Інформація про директорії, що моніторяться

Список директорій, що моніторяться, тобто належать до музичного архіву, також зберігатимуться у файлі та використовуватимуться при запуску програми для ініціалізації модуля, що відповідатиме за спостереженням за цими директоріями. Назвемо цей файл «monitiredFolders.xml». Структура файлу та візуальна схема будуть наведені нижче, у підпункті «Структура масивів інформації».

4) Список усіх доступних плейлистів

Інформація про всі доступні плейлисти: їх назва, список пісень, що до них належать, тощо, також зберігатиметься в XML-файлі. Дані використовуватимуться при запуску застосунку для ініціалізації модуля, що відповідатиме за менеджмент плейлистів. Назвемо цей файл «playlistManagerInfo.xml». Структура файлу та візуальна схема будуть наведені нижче, у підпункті «Структура масивів інформації».

5) Таймінг для плейлистів

Сам по собі плейлист являє лише набір пісень та їх послідовність виконання. Вже сукупність плейлистів та їх порядок створюють дещо вищу абстракцію – таймтейбл. Таймтейбл – це список плейлистів, порядок їх виконання, конктретна дата, абстракція, якою власне і

керуватиме музичний редактор. Таймтейбл містить в собі інформацію про те, які плейлисти звучатимуть в ефірі сьогодні чи навіть через рік. Ця інформація в жодному випадку не може бути втраченою при перезапуску застосунку. Тому її ми також зберігатимемо у файлі та прийматимемо в якісті вхідних даних на етапі запуску застосунку. Назвемо цей файл «playlistHistory.xml». Структура файлу та візуальна схема будуть наведені нижче, у підпункті «Структура масивів інформації».

2.2 Вихідні дані

На виході ми отримуємо:

– оновлені xml-файли, що служать водночас вхідними даними при запуску застосунку (це інформація про адресатів, плейлисти, таймінг плейлистів, інформація про всі пісні музичного архіву та директорії, що до нього належать);

– нові пісні у музичному архіві;

– листи на електронні адреси, підписані на оновлення.

Наведемо опис вихідних файлів у таблиці 2.1.

Таблиця 2.1 – Опис вихідних файлів

№	Найменування	Дані
1	Інформація про пісні у базі	- шлях до файлу - назва виконавця - назва пісні - тип пісні (для вечірки, нова, старий хіт, кастомний, тощо) - тривалість треку - унікальне ID пісні

Продовження таблиці 2.1

2	Плейлисти	- список пісень - тривалість плей-листа - унікальне ID плейлиста - назва плейлиста
3	Таймінг плейлистів	- унікальне ID плейлиста - запланована дата - запланований час початку
4	Інформація для email-нотифікацій	- текст email'у - імена та пошти адресатів
5	Директорії, що належать до музичного архіву	- список директорій, що підлягають моніторингу

2.3 Структура масивів інформації

Приклад1 – структура файлу songManager.xml

```
<songbase>
    <song>
        <type>5</type>
        <id>ft$3@!!kji78yTR0</id>
        <title>Skibidi</title>
        <author>Big Russian Boss</author>
        <filepath>C:\Users\nstupynets\Skibidi-
BigRussianBoss.mp3</filepath>
        <durationsec>362</durationsec>
    </song>
    <song>
```

```
        <type>8</type>
        <id>gv899@#kjAS8ydS9</id>
        <title>Crawling</title>
        <author>Linkin Park</author>
        <filepath>C:\Users\nstupynets\Crawling-
LinkinPark.mp3</filepath>
        <durationsec>289</durationsec>
    </song>
    ...
</songbase>
```

Структурна схема даного XML-файлу зображена на рисунку 2.1

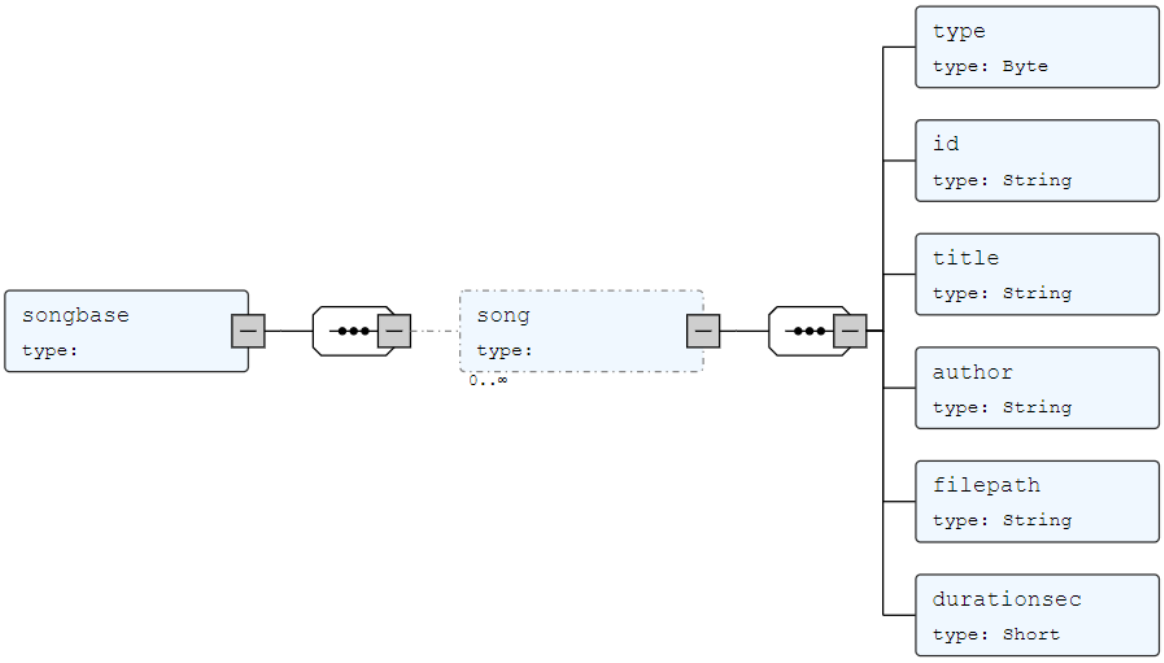


Рисунок 2.1 – Структурна схема songManager.xml XML-файлу

Приклад2 – структура файлу emailNotifierData.xml

```
<emailnotifier>
    <emailtextbegin>Please, check following list of
            recently added songs:</emailtextbegin>
    <emailtextend>Best regards, music
                editor</emailtextend>
    <receiverslist>
        <receiver>
            <name>Anna Ivanova</name>
            <emailaddres>anna.ivanova@gmail.com
                        </emailaddres>
        </receiver>
        <receiver>
            <name>Nataliia Bogdanova</name>
            <emailaddres>n.bogdanova@gmail.com
                        </emailaddres>
        </receiver>
        ...
    </receiverslist>
</emailnotifier>
```

Структурна схема даного XML-файлу зображена на рисунку 2.2

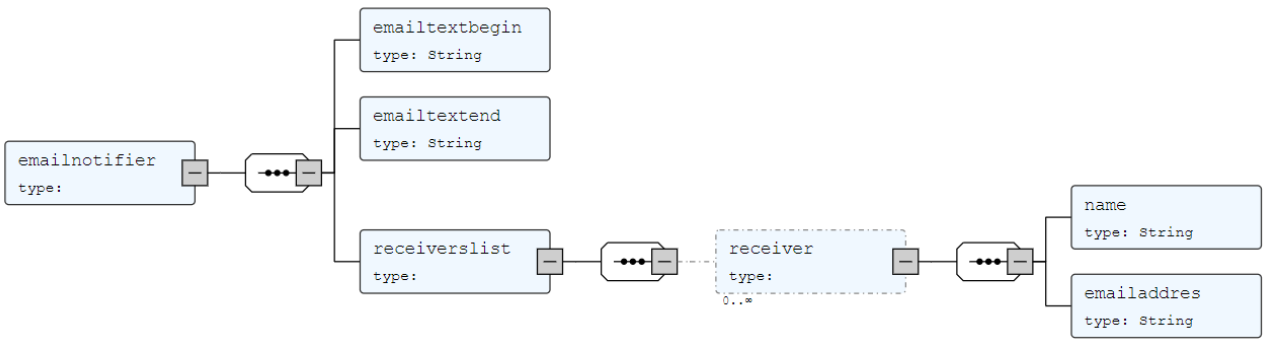


Рисунок 2.2 – Структурна схема emailNotifierData.xml XML-файлу

Приклад3 – структура файлу monitiredFolders.xml

```
<dirlist>
    <dir path="C:\Users\nst\DIPLOMA\songs"></dir>
    <dir path="C:\Users\nstupynets\music"></dir>
    ...
</dirlist>
```

Схема даного XML-файлу найпростіша з усіх, адже як можемо побачити з прикладу, фактично, являє собою лише список директорій. Структурна схема для цього XML-файлу зображена на рисунку 2.3

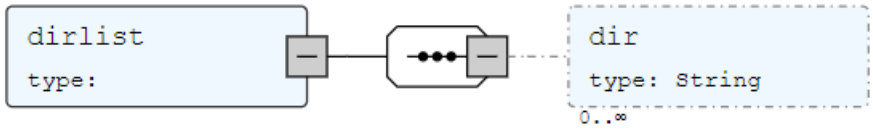


Рисунок 2.3 – Структурна схема monitoredFolders.xml XML-файлу

Приклад4 – структура файлу playlistManagerInfo.xml

```
<playlistitems>
    <playlist>
        <id>gT911@#kjDF8ydS9</id>
        <name>myPlayList1</name>>
        <durationsec>3600</durationsec>
        <songlist>
            <song>
                <id>ft$3@#!kji78yTS5</id>
                <order>0</order>
            </song>
            <song>
                <id>ft!2@#tuT?78yTS5</id>
                <order>1</order>
            </song>
            ...
```

```
            </songlist>
        </playlist>
    </playlistitems>
```

Структурна схема даного XML-файлу зображена на рисунку 2.4

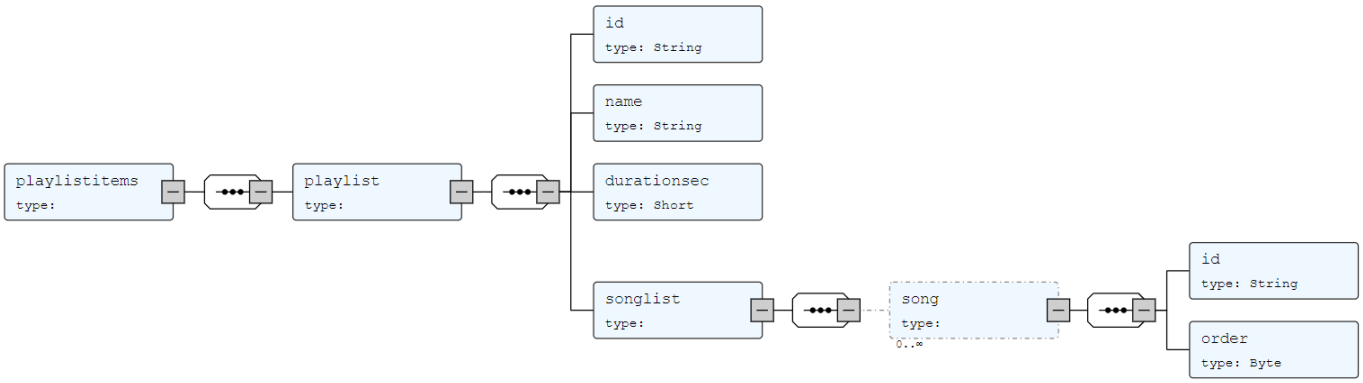


Рисунок 2.4 – Структурна схема playlistManagerInfo.xml XML-файлу

Приклад5 – структура файлу playlistHistory.xml

```
<playlisthistory>
    <historyitem>
        <playlistid>ft$3@#!kji78yTS5</playlistid>
        <scheduleddate>10-06-2019</scheduleddate>
        <starttime>10:50:54</starttime>
    </historyitem>
    <historyitem>
        <playlistid>ft!2@#tuT?78yTS5</playlistid>
        <scheduleddate>10-06-2019</scheduleddate>
        <starttime>11:50:54</starttime>
    </historyitem>
    ...
</playlisthistory>
```

Структурна схема даного XML-файлу зображена на рисунку 2.5

Рисунок 2.5 – Структурна схема playlistHistory.xml XML-файлу

Висновок до розділу

Отже, ми описали всі необхідні застосунку дані, те для чого нам потрібні ці вхідні дані, як ми зможемо їх використати та що отримаємо на виході. Також у даному розділі була наведена детальна інформація про структуру вхідних даних та зображена їх схема, підкріплена прикладами.

3 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Засоби розробки

Даний застосунок для автоматизації робочоо місця музичного редактора було розроблено за допомогою мови програмування C++.

Перевагами даної мови програмування є її універсальність, багатофункціональність та платформонезалежність. Дана мова програмування може бути використана для розроблення додатків на операційних системах Windows, Mac OS, та на всіх дистрибутивах Linux. Основною перевагою мови програмування C++ є можливість безпосередньої роботи з пам’яттю процеса (запозичено від мови програмування C) та водночас об’єктно орієнтований підхід, що відрізняє мову програмування C++ від C та робить її більш високорівневою.

Саме об’єктно орієнтований підхід був обраний до вирішування даної задачі. Навколишній світ змушує нас змінювати свої рішення під впливом якихось обставин, про котрі ми не можемо знати наперед. Саме це лежить в основі об’єктно орієнтованого підходу – пристосовування програмного забезпечення до різноманітних обставин, котрі можуть повпливати на його функціонування, навчити додаток відповідно реагувати на різноманітні ситуації.

Для функціонування мережевої частини додатку було обрано протокол TCP. Переваги цього протоколу над протоколом UDP у тому, що обидві сторони мережевого функціоналу повідомлені про те, чи їхні дані, котрі надсилалися через мережу, були успішно доставлені до вказаного місця, чи ні і їх потрібно надсилати повторно. Оскільки в даному додатку важливим є правильне надсилання медіафайлів у повному їх обсязі, користувач повинен мати гарантії того, що його дані не загубляться у мережі і будуть доставлені за призначенням. Для реалізації мережевого функціоналу було обрано безкоштовну бібліотеку boost.

Бібліотека boost це відома бібліотека серед C++ розробників. Вона славиться своїм майже всеохоплюючим функціоналом. Використовуючи дану бібліотеку, можна працювати зі всіма надбудовами над функціоналом, котрий описаний стандартом C++, використовувати обгортки системних функцій, що дозволяють працювати з реєстрами, файловою системою, мережею, об'єктами ядра і і ще багато чим. Важливою властивістю цієї бібліотеки є те, що вона є платформонезалежною (тобто окрім обгорток над мовним функціоналом, який і так є платформонезалежним, обгортки над системними функціями теж передбачають свою реалізацію для різних систем).

Написаний код на мові програмування C++ компілювався офіційним компілятором від компанії Microsoft msvc2017 64bit.

Для розроблення користувацького інтерфейсу було обрано безкоштовну IDE Qt. Основні переваги Qt в тому, що це IDE, котра має свої візуальні компоненти і орієнтована для програмування на C++. Бібліотека Qt наповнена безліччю корисного функціоналу, що, в свою чергу, спрощує розробку та інтеграцію написаного коду з всіма візуальними компонентами. Qt серед розробників славиться своїм унікальним механізмом сигналів та слотів (даний механізм є своєрідною реалізацією патерну Observer – один до всіх і вс до одного). Всі візуальні компоненти, котрі є на вікнах, і самі вікна, будуються в дерева відношень. Це в подальшому дозволяє всім елементами комунікувати між собою. Даний функціонал є дуже важливим та необхідним під час розробки якісного програмного забезпечення.

Слід зазначити, що основна частина додатку, що має користувацький інтерфейс, водночас є серверною частиною мережевого функціоналу. По той бік мережі, клієнт, являє собою програму без користувацького інтерфейсу. Основною функцією клієнта є відтворення аудіозаписів, що були обрані серверною частиною та відправлені нею до клієнта.

Функціонал Клієнта є платформозалежним та наразі виконується на платформі Windows. Для розробки клієнтської частини було обрано IDE Microsoft Visual Studio 2017.

3.2 Вимоги до технічного забезпечення

3.2.1 Загальні вимоги

Будь-яка радіостанція, котра зацікавлена в даному програмному забезпеченні, вже має комп’ютери, що слугують серверами для збереження аудіофайлів.

Щодня музичний редактор повинен фільтрувати, завантажувати, сортувати, видаляти та проводити інші операції з аудіофайлами перед тим, як вони потраплять в ефір. Для того, щоб локальну обчислювальну машину можна було використовувати для роботи з даним застосунком, вона повинна бути оснащена :

- процесором з тактовою частотою 1,3 ГГц і більше;
- базовою відеокартою (або інтегрованою) ;
- базовою звуковою картою;
- мережевою картою;
- оперативною пам’яттю у розмірі 512 МБайт, чи більше;
- операційною системою Windows XP, чи новішою;
- сервер і клієнт повинні заходитись в одній підмережі;
- сервер повинен мати доступ до мережі інтернет зі швидкістю не менше 1 Мбіт/секунду;
- сервер повинен мати пристрої користувацького вводу – клавіатуру та маніпулятор, а також пристрій відображення – монітор;
- клієнт повинен додатково мати встановлений Windows Media Player та бути оснащеним гарнітурою.

3.3 Архітектура програмного забезпечення

Доцільною буде побудова діаграм наведених у таблиці 3.1.

Таблиця 3.1 – Тип наведених діаграм та їх роль процесі проектування

Тип діаграми	*Роль у процесі проектування*
Діаграма послідовності	Визначає набір класів та їх послідовність взаємодії
Діаграма класів	Визначає набір класів та їх взаємодію
Діаграма компонентів	Визначає групування класів у компоненти та ланцюжки виклику компонентів іншими компонентами протягом роботи застосунку.

3.3.1 Особливості використання діаграми послідовності

Діаграма послідовності є засобом визначення послідовності взаємодії класів для конкретного випадку. Наприклад, який клас звертатиметься до якого при виконанні певної дії. Маючи розгорнуту діаграму послідовностей можна прослідкувати весь ланцюжок викликів, що відбуватимуться на «внутрішньому рівні», тобто рівні класів, для виконання певної функції застосунку. Також варто зазначити, що діграма послідовностей належить до так званих «*Behavioral UML diagrams*»[3]. Тобто, акцент тут робиться не на організації структур даних, а на їх поведінці – взаємодії.

3.3.2 Особливості використання діаграми класів

Грамотно реалізована діаграма класів – це один з факторів якісного програмного забезпечення. На практиці, перш ніж приступати до реалізації певного функціоналу, обов’язково повинна бути реалізована саме діаграма класів. Вона описує набір необхідних класів та їх звязки між собою. Діаграма класів дає чіткі відповіді на запитання, хто кого наслідуватиме, хто у кому агрегуватиметься, тобто міститиметься, які класи будуть взаємодіяти між

собою, тобто перебуватимуть в асоціації. Також вона допомагає побачити проблемні ділянки, прогалини в архітектурі, які легко можуть залишитися непоміченими її відсутності. Скажімо, лічені секунди займе виявлення *Діамантової проблеми(Ромбовидне наслідування)* при наслідуванні, абощо.

3.3.3 Особливості використання діаграми компонентів

Діаграма компонентів є структурною UML діаграмою, так само як і діаграма класів. Тобто головними тут є саме структури даних. Але якщо в діаграмі класів ми «опускаємося» до рівня кожного класу. То при створення діаграми компонентів ми об’єднуємо ці класи та формуємо з них компоненти, взаємодію між якими згодом описуємо. Тобто вона дає змогу поглянути на архітектуру застосунку не вдаючися до найдрібніших побробдиць, як це робиться в діаграмі класів.

3.4 Приклади

3.4.1 Діаграма послідовності

Наведемо схеми структурні послідовності для наступних випадків:

- ініціалізація класів при запуску застосонку;
- пошук нових пісень у мережі;
- додавання пісень до музичної бази;
- оновлення користувацького інтерфейсу внаслідок комунікації між клієнтом та сервером;
- додавання нового плейлиста.

Схема структурана послідовності ініціалізації класів при запуску застосонку наведена у графічному матеріалі [3].

При пошуку нових пісень серед мережевих ресурсів комунікація класів виглядатиме абсолютно інакше. У графічному матеріалі [4] зображена схема структурна послідовності пошуку нових пісень в мережі.

Під додаванням пісень до музичної бази мається на увазі додавання пісень у музичний архів без участі нашого застосунку, тобто шляхом переміщення файлів в одну з директорій, що моніториться. Дана схема наведена на рисунку 3.1

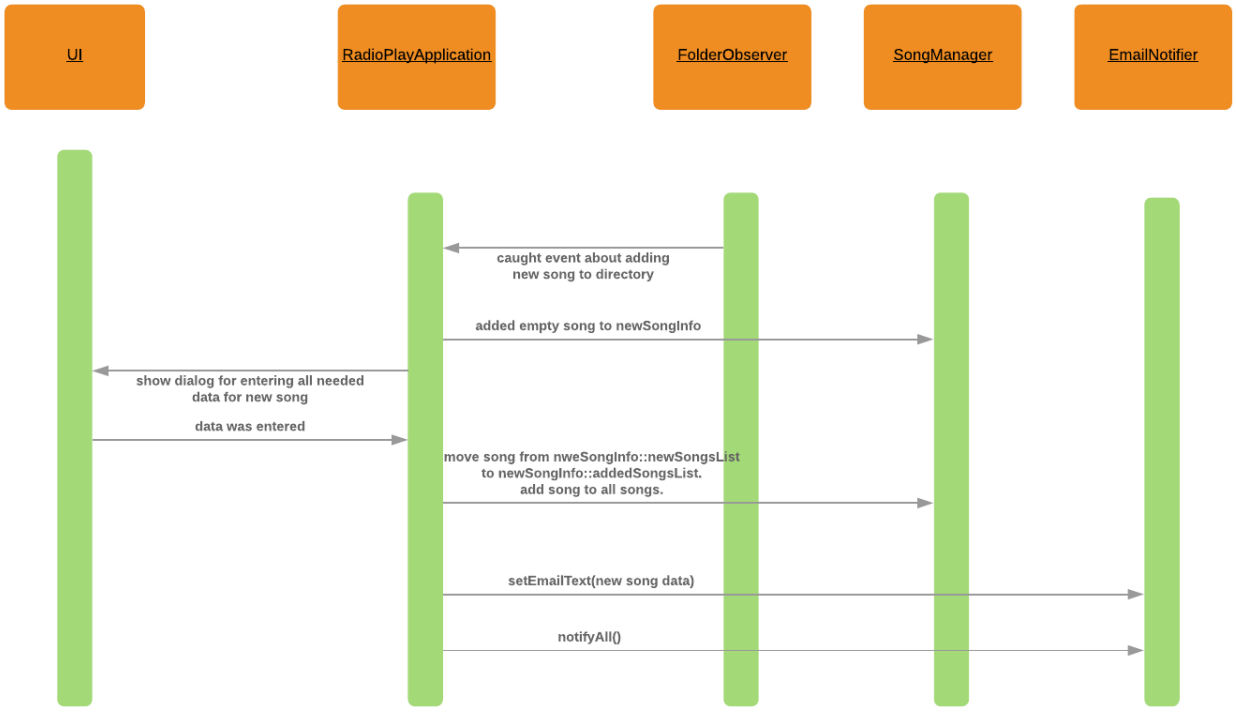


Рисунок 3.1 – Схема структурна послідовності додавання пісень до музичної бази

Схема структурна послідовності оновлення користувацького інтерфейсу внаслідок комунікації між клієнтом та сервером наведена у графічному матеріалі [5]. Користувач генерує плейлист та додає його до розкладу виконання плейлистів. Клієнтська частина відповідає за відтворення композицій, що належать до цього плейлиста, але окрім цього також інформує серверну частину про те, який плейлист виконується та яка саме композиція звучить на даний момент. Тобто користувач так званого

основного застосунку має змогу спостерігати за послідовністю відтворень композицій та плейлистів, що відбувається на стороні клієнта. Ця інформація відображатиметься на користувацькому інтерфейсі.

Наведемо також схему структурну послідовності додавання згенерованого плейлиста до розкладу відтворення плейлистів на задану дату та час у графічному матеріалі [6].

3.4.2 Діаграма класів

Схема структурна класів програмного забезпечення наведена у графічному матеріалі [2].

3.4.3 Діаграма компонентів

Схема структурна компонентів зображена на рисунку 3.2

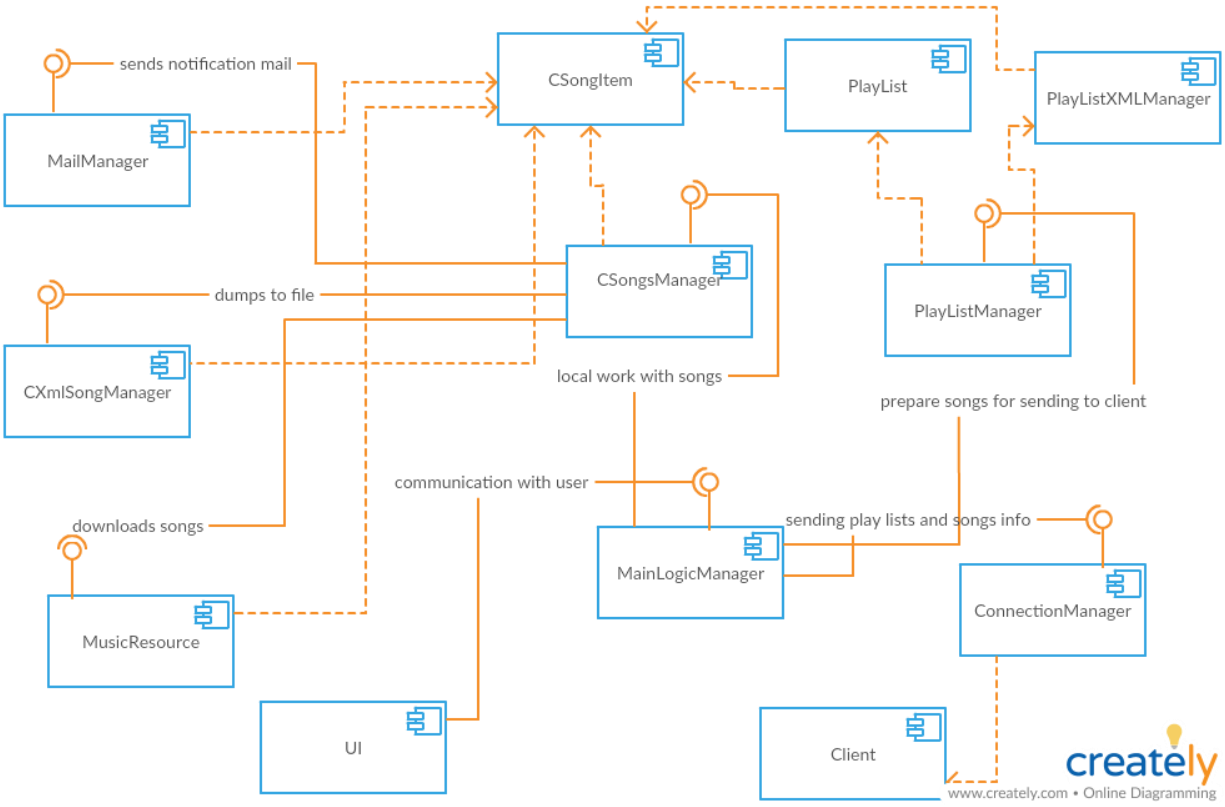


Рисунок 3.2 – Схема структурна компонентів

3.4.4 Специфікація функцій

Таблиця 3.2 - Функції класів програмного забезпечення

Назва	Примітка
RadioPlayApplication	Основний клас у нашому застосунку. Він забезпечуватиме комунікацію класів-менеджерів. Якщо щось зміниться в одному класі, то він нотифікує про це даний клас, а даних клас знає хто саме підписаний на ці зміни і нотифікує ті класи.
XMLParserBase	Абстрактний клас від якого наслідуватимуться всі XML парсери.
SongManagerXMLParser	Клас, що відповідатиме за парсинг XML-ки з даними для ініціалізації SongManager класу на стартапі.
PlaylistManagerXMLParser	Клас, що відповідатиме за парсинг XML-ки з даними для ініціалізації PlaylistManager класу на стартапі.
PlaylistHistoryXMLParser	Клас, що відповідатиме за парсинг XML-ки з даними для ініціалізації PlaylistHistory класу на стартапі.
EmailInfoXMLParser	Клас, що відповідатиме за парсинг XML-ки з даними для ініціалізації EmailNotifier класу на стартапі.
MonitoredFoldersXMLParser	Клас, що відповідатиме за парсинг XML-ки з даними для ініціалізації FolderObserver класу на стартапі.

Продовження таблиці 3.2

XMLParserFactory	Клас, що відповідатиме за створення вищезгаданих парсерів. Якщо комусь потрібен буде парсер, то він не створюватиме його напряму, а звертатиметься до цього класу, щоб він або створив новий, або віддав вже існуючий.
SongManager	Клас, що міститиме в собі контейнери об‘єктів пісень та джинглів та оперуватиме ними. Якщо, скажімо, потрібно буде додати нову пісню – то це саме той клас, якого ми проситимемо це зробити.
AbstractAudioResource	Абстрактний клас, від якого наслідуються класи Jingle та Song. Однакові для цих обох класів члени винесені до даного абстрактного класу. Також був використаний для того, щоб показати спорідненість цих двох класів.
Song	Клас, що зберігатиме в собі інформацію про конкретну пісню: її назву, автора, тип, тривалість, шлях до неї. Саме контейнером об‘єктів цього класу оперуватиме SongManager.
Jingle	Клас, що описує реальний джингл (коротка вставка між піснями). Відрізняється від пісні тим, що у нього немає типу, автора, тощо. Тобто, це більш примітивний тип. Також джингли, на відміну від пісень, можуть повторюватися протягом однієї години.

Продовження таблиці 3.2

NewSongsInfo	Допоміжна структура даних, що міститиме в собі інформацію про пісні, які перебувають у «висячому» стані, тобто нові пісні, що були скачані в результаті пошуку нової музики по музичних інтернет-ресурсах, але користувач ще не вирішив чи він додає їх до музичного архіву чи ні.
Playlist	Клас, що міститиме в собі інформацію про один конкретний плейлист: тривалість, список пісень, шаблон, за яким його було згенеровано, тощо.
PlaylistConfig	Невеликий допоміжний клас, що міститиме в собі стандартні параметри необхідні для генерування плейлиста. Користувач може визначити цей конфіг один раз і згодом лише застосовувати його для генерування, не налаштовуючи конфігурацію повторно. Містить в собі інформацію про тривалість плейлиста та про кількість пісень на джингл (тобто скільки пісень має програти перш ніж прозвучить джингл).
PlaylistGenerator	Клас, що відповідає за генерування плейлистів. Ми надаємо йому конфігурацію та шаблон, а він нам – готовий плейлист.

Продовження таблиці 3.2

PlaylistHistory	Клас, що відповідає за таймінг виконання плейлистів. Містить в собі мапу, що складається з id плейлиста та час початку його виконання.
PlaylistManager	Клас, що міститиме в собі контейнери об‘єктів плелистів та оперуватиме ними. Якщо, скажімо, потрібно буде додати ти видалити попередньо згенерований плейлист – то це саме той клас, якого ми проситимемо це зробити.
PlaylistPattern	Структура, що міститиме в собі інформацію необхідну для генерації плейлиста: які типи пісень повинні бути присутніми у плейлисті, а які – ні, тощо.
TCPServer	Відповідає за комунікацію з клієнтом, обмінюється з ним повідомленнями. Надсилає клієнту плейлист, що зберігається в PlaylistHistory згідно з таймінгом, а також самі пісні.
NetworkManager	Містить в собі TCPServer, що відповідає щза комунікацію з клієнтом. Окрім цього відповідає за все, що пов’язане, що мережею. Якщо музичний архів знаходить посилання на нові пісні, то він надає їх саме цьому класу з проханням їх скачати, тощо.
MusicArchivesManager	Відповідає за менеджмент музичних ресурсів, містить в собі інформацію про те, які музичні ресурси будуть використовуватися для пошуку нових пісень, а які – ні.

Продовження таблиці 3.2

IMusicArchive	Інтерфейс для узагальннення всіх музичних архівів. Наразі доступний лише один музичний архів, що від нього наслідується. MusicArchivesManager оперує саме референсами на цей інтерфейс, а не конкретними реалізаціїями.
FreeMusicArchive	Музичний архів, що представляє реальний ресурс у мережі, з якого ми скачуємо найновіші пісні. Цей клас «знає» яким чиному ми будемо комунікувати з цим ресурсом, які посилання, тощо нам для цього потрібні.
FolderObserver	Клас, що відповідатиме за моніторинг директорій, які належать до музичної бази.
EmailNotifier	Клас, що відповідатиме за сповіщення співробітників шляхом надсилання їм листів на робочу пошту.
UIComponent	Клас, що містить інформацію про всі юайні компоненти та виконує функцію схожу до функції RadioPlayApplication – тобто забезпечує комунікацію UI-компонент між собою.

Окрім класів, що описують базову бізнес-логіку проекту, наведемо також таблицю 3.3, де описані класи та їх функції для UI-компонент. В нашому випадку для кожного головного діалогу було створено клас.

Таблиця 3.3 – Функції UI-класів програмного забезпечення

MainWindow	Клас, що описує поведінку основного вікна застосунку. Містить в собі список пісень, що відтворюються та статус про те, яка пісня грає на даний момент. Окрім цього містить таймінг плей-листів та інформацію про них. Також доступна можливість редагування таймінгу плейлистів, тощо. Саме цей діалог доступний одразу після запуску застосунку. Окрім цього містить у собі Toolbar, в якому доступні різноманітні опції, як наприклад запуск пошуку нової музики, редагування патернів плейлистів, налаштування Gmail-нотифікацій, тощо. Результатом вибору цих опцій буде відкривання новогу діалогу, що відповідає за той чи інший функціонал. Але саме відкривання звійснюватиме клас UIComponent, MainWindow лише інформуватиме цю компоненту про певні дії зі сторони користувача. Саме UIComponent вирішуватиме як реагувати на ці дії.
AddPatternDialog	Діалог для створення нових патернів плейлистів. Для того, щоб створити новий плейлист, користувачу буде необхідно ввести назву нового плейлиста та обрати які теги будуть задовільняти даний патерн, а які мають бути виключені.

Продовження таблиці 3.3

DialogAbout	Клас, що відповідає за діалог, який містить інформація про застосунок та автора. Залежно від того, яку опцію обирає користувач, у діалозі відображатиметься або інформація про застосунок, або інформація про автора програми. Користувач може відкрити даний діалог за допомогою вибору відповідного пункту меню, що доступне у головному діалозі.
GenerateNewPlaylistDialog	Діалог, що відповідатиме за генерацію нових плейлистів. Для того, щоб згенерувати новий плейлист, необхідно буде обрати наявний патерн, а також конфігурацію (тривалість плейлиста, кількість пісень на один джингл). Доступна можливість використання дефолтної конфігурації.
GmailOptionsDialog	В даному діалозі доступна конфігурація Gmail-нотифікацій. Можливість редагування шаблону листів до користувачів, логін та пароль музичного редактора до сервісу Gmail, тощо.
ManageSubscribersDialog	Діалог, у якому доступний список поточних підписників на зміни у музичній базі пісень та можливість редагувати список цих підписників.

Продовження таблиці 3.3

ManageMusicArchiveDialog	Діалог, що відповідає за керування музичним архівом. Тут доступні можливості додавання та видалення пісень, заповнення інформації про щойно додані пісні, тощо.
DialogMusicResources	Діалог керування музичними ресурсами. Тут можна обрати, які музичні інтернет-ресурси використовуватимуться для пошуку нових пісень, а які – ні.
DialogSearchNewMusic	Діалог, що відповідає за запуск пошуку нової музики. Коли в тулбарі користувач обирає пунк ‘Search New Music’, відкривається даний діалог, у ньому відображатиметься список ресурсів, що використовуватимуться для пошуку нової музики. Якщо кількість ресурсів більша або рівна 1, то буде доступна кнопка ‘Start search’, натискання якої запускає пошук нових пісень у мережі.
DialogWaitForMusicSearch	Діалог, що відображатиметься коли процес пошуку нової музики триватиме. Доступна можливість зупинити пошук.
MusicArciveDirs	Діалог, що відповідає за керування директоріями, що належать до музичного архіву. Тут доступні можливості додавання та видалення директорій. Саме ці директорії і моніторяться класом DirectoryObserver.

Продовження таблиці 3.3

PlaylistPatternsDialog	Діалог, що відповідає за керування доступними плейлистами. Доступна опція додавання нового, видалення вже існуючого патерна, а також їх редагування.
RecentlyDownloadedSongsDialog	Після того, як нові пісні були скачані, користувач повинен обрати які з цих пісень додавати до музичного архуву, а які – ні. Дану можливість надаватиме саме цей діалог. Також доступна опція прослуховування пісень.

Висновок до розділу

В даному розділі ми сформулювали основні вимоги до технічного забезпечення для того, щоб наш застосунок міг стабільно працювати. Також описали технології, що використовувалися для розробки програмного забезпечення та обгрунтували вибір саме цих технологій. Також була описана архітетура застосунку, що мамиме неабияку користь для його можливої подальшої підтримки та покращення, збільшення функціональності, тощо. Наведені діаграми допоможуть іншим людям ознайомитися зі структурою та поведінкою проекту не витрачаючи годин на читання коду.

4 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

4.1 Керівництво користувача

На рисунку 4.1 зображене головне меню програми, на головному вікні можна також побачити список пісень, що відтворюються на даний момент зі сторони клієнта. Окрім цього на головному вікні доступні основні сповіщення, які потребують певних дій зі сторони користувача, та кнопки при натисканні яких такі дії або будуть виконані одразу, або відкриється нове вікно в якому цю дію можна буде завершити. На рисунку 4.1 бачимо, що таких сповіщень є два: «New songs were added to archive» та кнопка «Notify» (для виконання дії – сповіщення всіх підписників), а також «Fill info about new added songs» та кнопка «Add»(результатом натискання цієї кнопки буде відкриття нового вікна для введення інформації про нові пісні).

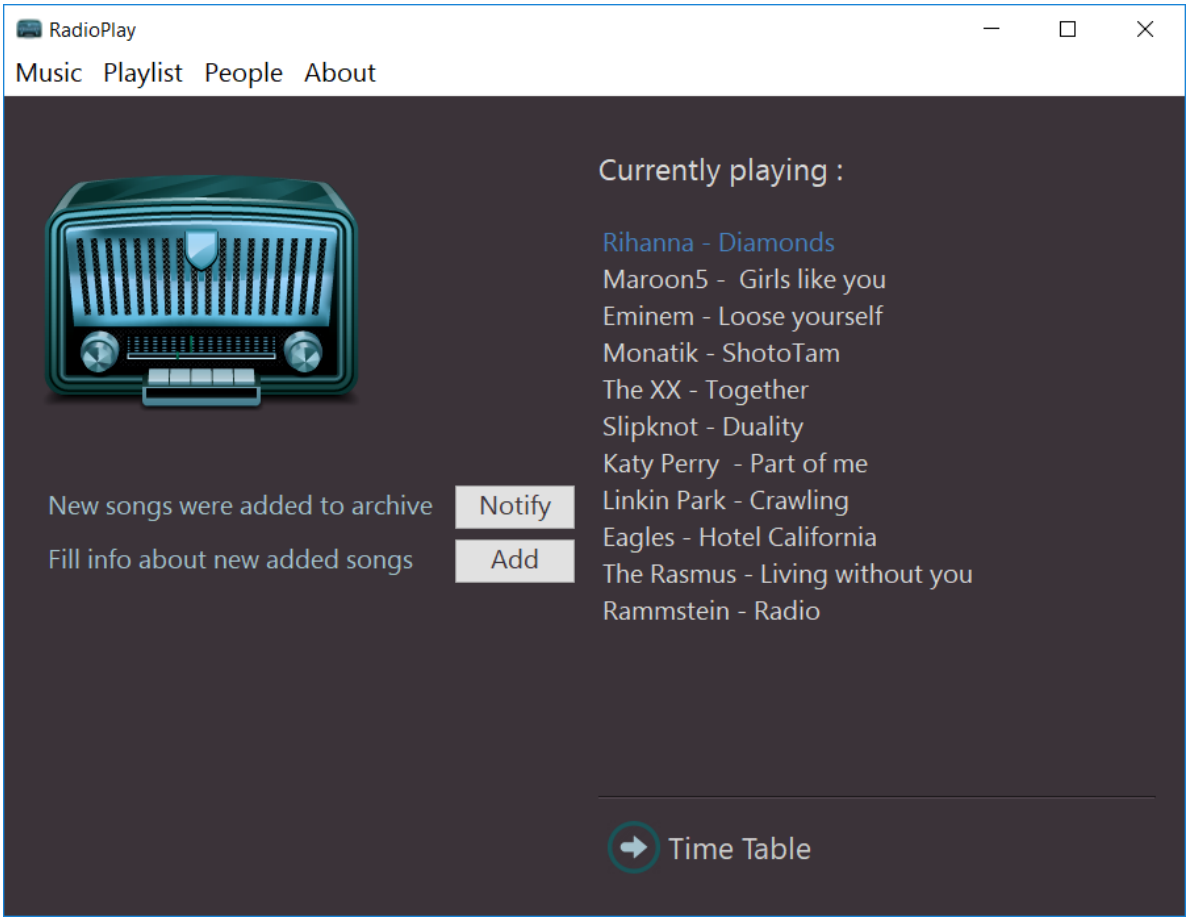


Рисунок 4.1 – Головне меню програми

Також доступна кнопка «Time Table», після натискання якої на головному вікні буде зображена більш детальна інформація про статус відтворення композицій та плейлистів на стороні клієнта. Вигляд форми «Time Table» зображено на рисунку 4.2

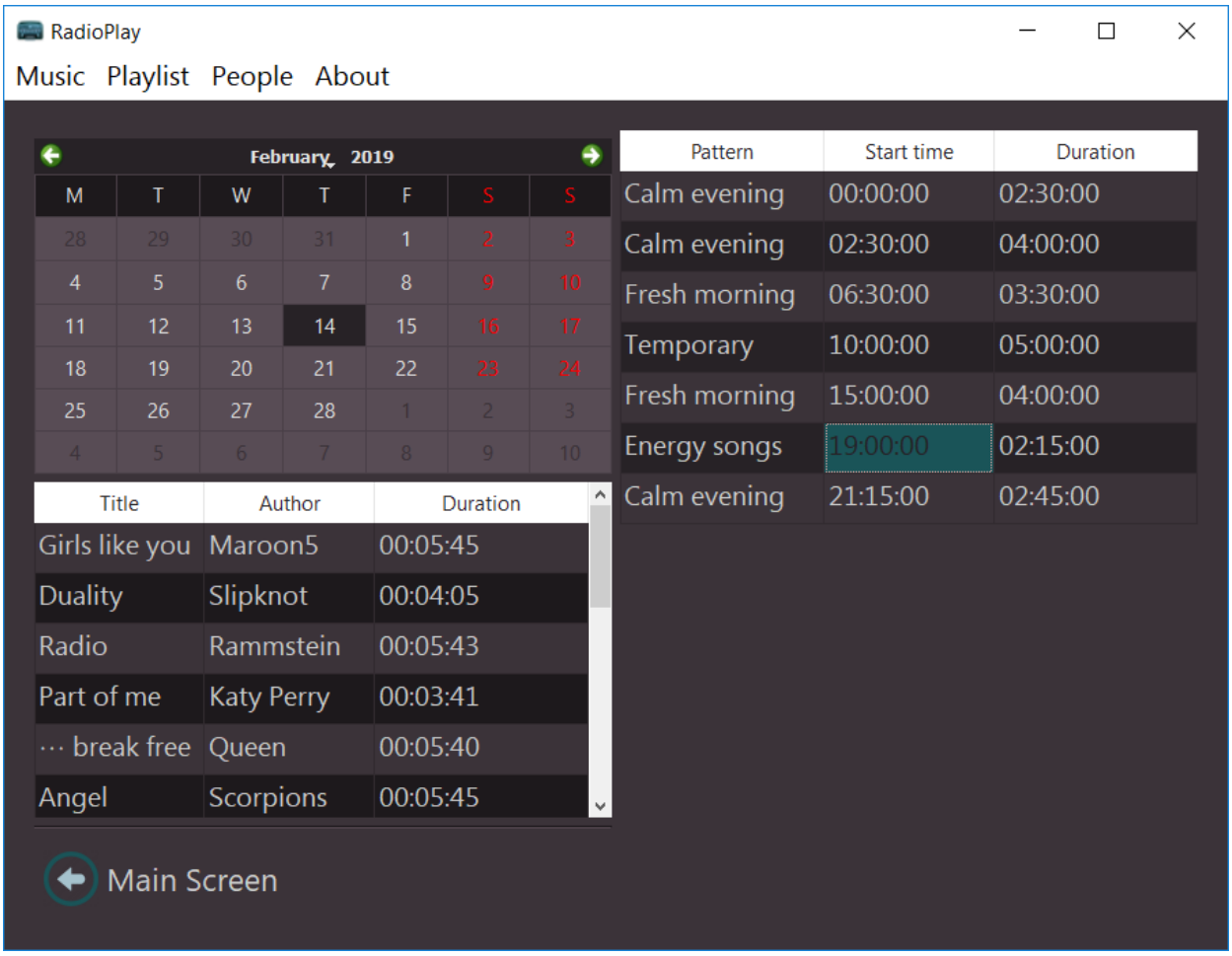


Рисунок 4.2 – Форма «Time Table» головного меню програми

Для того, щоб переглянути інформацію про розклад відтворення на певну дату, потрібно обрати цю дату в календарі, тоді у таблиці, що праворуч буде доступна інформація про плейлисти, що звучатимуть. У цьому списку можна обрати конкретний плейлист, тоді таблиця, що розташована під календарем, оновиться та відображатиме список пісень, що належать до цього плейлиста. Пісні посортовані у порядку свого виконання.

Для зручного керування музичним архівом в верхню панель головного вікна було винесено всі можливі дії, верхню панель розбито на декілька

логічних блоків. Список цих блоків та підпунктів меню, що до них належать, наведено нижче:

- Music (керування музикою):
    а) Search New (пошук нової музики);
    б) Manage Music Resources (керування інтернет-ресурсами для скачування нової музики);
    в) Manage Music Archive (керування музичним архівом);
    г) Music Archive Dirs (керування директоріями, що належать до музичного архіву);
- Playlist (керування плейлистами):
    а) Manage Playlist Patterns (керування шаблонами плей-листів);
    б) Generate Playlist (генерування нових плейлистів за шаблоном);
- People (керування підписниками):
    а) Gmail Options (налаштування Gmail-сповіщень)
    б) Subscribers (керування списом підписників)
- About (інформація):
    а) About RadioPlay (інформація про застосунок)
    б) About Author (інформація про автора)

Щоб розпочати пошук нової музики необхідно обрати пункт ‘Search new’. Після цього відкриється вікно «Search New Music», зображене на рисунку 4.3.

Рисунок 4.3 – Вікно пошуку нової музики

У даному вікні корисувач може побачити список ресурсів, по яких буде здійснюватися пошук. Тут бачимо, що пошук буде здійснюватися лише по ресурсу FreeMusicArchive. Якщо список буде пустий – кнопка «Start rearch» буде недоступна. Праворуч від списку доступна кнопка «Change resources», натиснувши на яку, користувач зможе змінити список доданих ресурсів.

Якщо користувач натисне кнопку «Change resources», то побачить нову вікно «Manage Music Resources», яке зображено на рисунку 4.4.

Рисунок 4.4 – Вікно менеджера музичних ресурсів

На даному етапі існування ПЗ доступний лише один ресурс, який користувач може або додати до пошуку, або виключити. Навіть після завершення програми та її нового старту дані про те, які ресурси додані, а які – ні, збережуться.

Повернемося до попереднього вікна та розпочнемо пошук пісень по ресурсах. Для цього потрібно натиснути кнопку «Start search». Після того, як ми це зробимо, вікно пошуку нової музики закриється та відкриється вікно очікування, його зображено на рисунку 4.5.

Рисунок 4.5 – Вікно очікування

Після того, як пошук буде завершено, дане вікно закриється та з’явиться нове вікно – з результатами пошуку (рисунок 4.6). В даному вікні користувач зможе прослухати музику та додати її до архіву, або ж видалити, якщо трек не підходить. Інформація про додані пісні буде подана у вигляді списку. Коли користувач натисне на елемент списку, він також зможе побачити додаткову інформацію про автора та назву пісні.

Рисунок 4.6 – Вікно з результатами пошуку

Повернемося до головного меню та оберемо пункт «Manage Archive Dirs». На екрані з’явиться вікно «Music Archive Dirs» (рисунок 4.7).

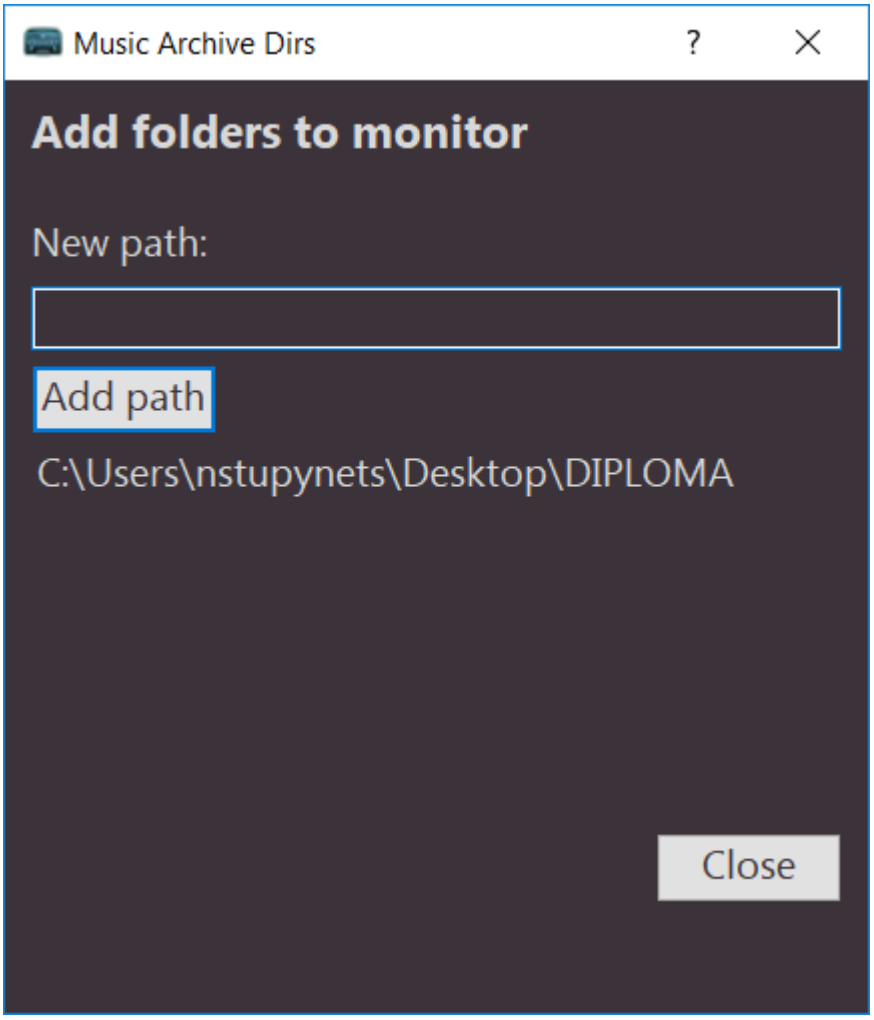


Рисунок 4.7 – Менеджер музичного архіву

Тут користувач може додати або видалити директорії з піснями, що будуть моніторитися. Для того, щоб додати шлях, достатньо надрукувати його у виділеному полі для текстового вводу та натиснути кнопку «Add path». Якщо введений шлях вже буде доданий, або буде таким, що не існує, то ми побачимо відповідні повідомлення про це, зображені на рисунках 4.8 та 4.9

Для того щоб видалити директорію, потрібно двічі клікнути на елемент. Тоді появиться вікно, у якому користувача буде запитано чи він справді хоче видалити цей елемент. Вікно з підтвердженням зображене на рисунку 4.10. Для того, щоб елемент видалився, достатньо натиснути кнопку

«Yei», якщо користувач передумав чи помилково двічі клікнув не на той елемент, то він може скасувати видалення, натиснувши кнопку «No».

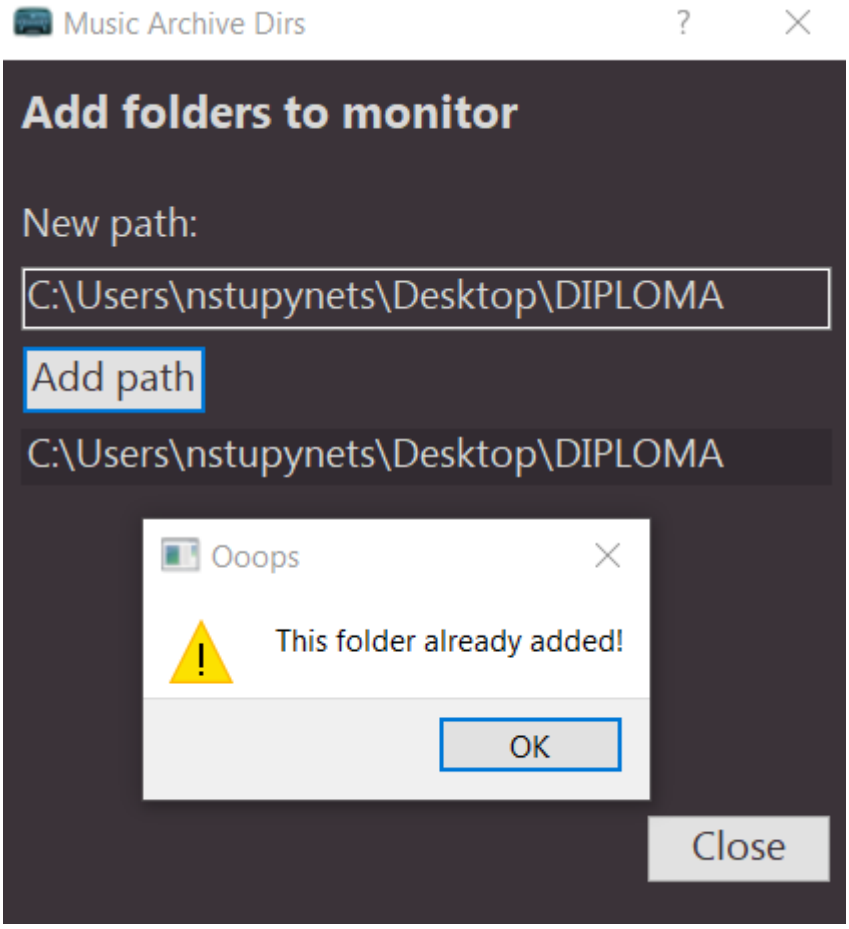


Рисунок 4.8 – Повідомлення про помилку, якщо вказаний шлях вже був попередньо доданий

Рисунок 4.9 – Повідомлення про помилку, якщо вказаний шлях не вірний

Рисунок 4.10 – Запитуємо користувача чи він справді хоче видалити елемент

Ще один із пунктів головного меню – менеджер шаблонів плейлистів (Manage Playlist Patterns). Якщо ми оберемо цей елемент то побачимо наступне вікно (рисунок 4.11).

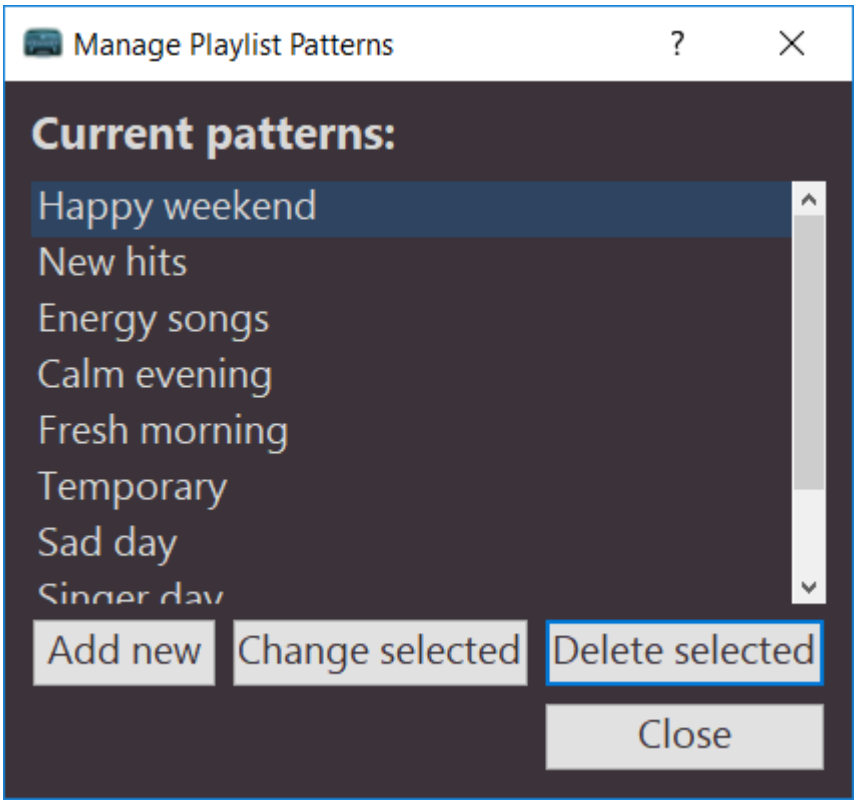


Рисунок 4.11 – Керування шаблонами плейлистів

Тут можна видалити чи відредагувати вже існуючий шаблон плейлиста. Також є можливість додати новий шабон. Для того, щоб це зробити, необхідно натиснути кнопку «Add new». Результатом натискання цієї кнопки буде відкриття вікня для додавання нового плейлиста, це вікно зображено на рисунку 4.12.

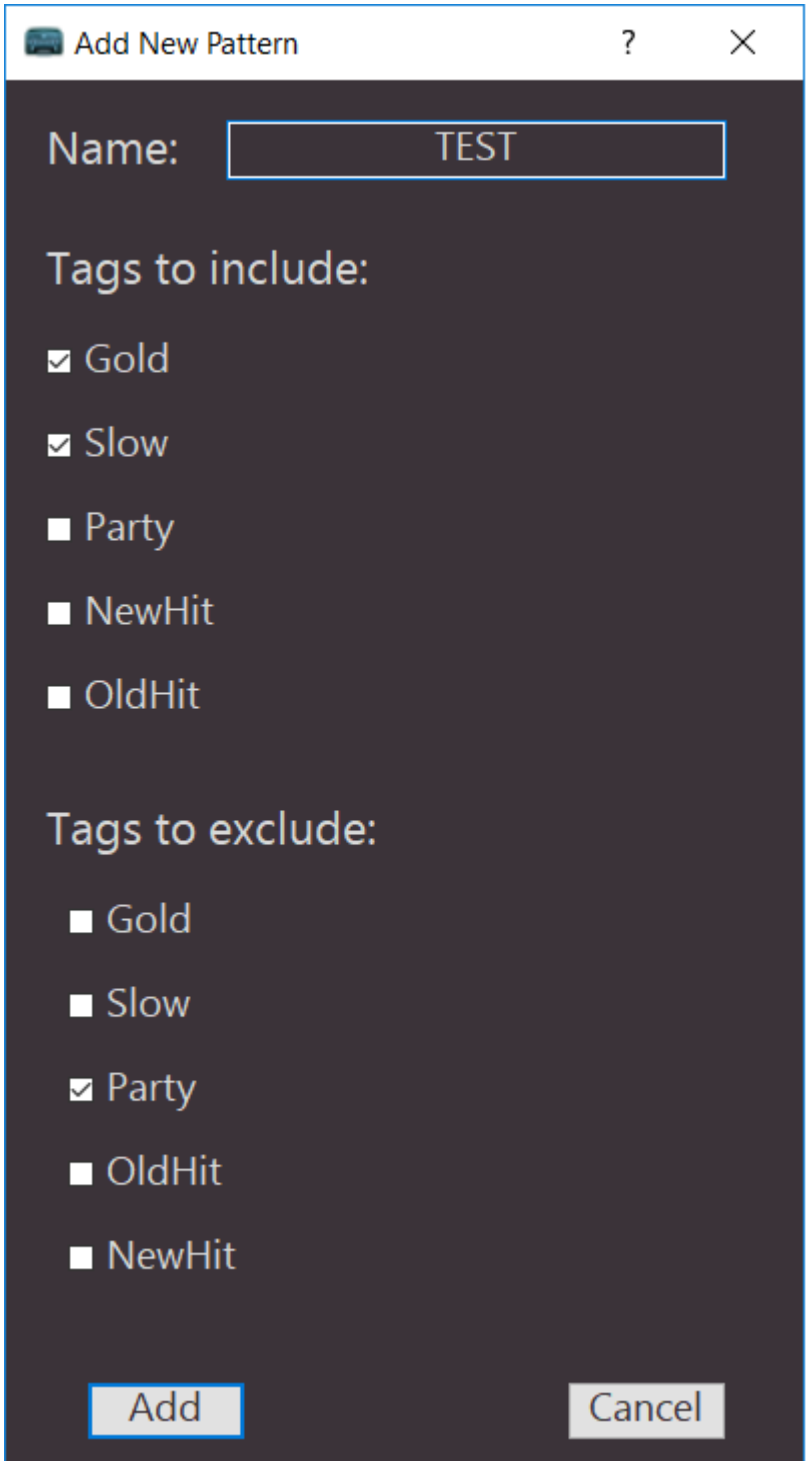


Рисунок 4.12 – Додавання нового шаблону плейлистів

На останок, користувачу доступна інформація про програму та про її автора. Для того, щоб її переглянути, необхідно обрати пункти меню «About RadioPlay» або «About Author» відповідно. Результати зображено на рисунках 4.13 та 4.14.

Рисунок 4.13 – Інформація про застосунок

Рисунок 4.14 – Інформація про автора

4.2 Методика випробувань програмного продукту

В даному розділі ми опишемо вибробування, які проводилися для того, щоб перевірити чи програмний продукт відповідає вимогам та поводитиме себе коректно в залежності від різних обставин.

Найголовніше правило, яке повинне виконуватися для будь-якого програмного забезпечення – застосунок не повинен екстренно завершувати свою роботу за жодних умов. Таких проблем не повинно виникати взагалі, адже коли користувач бачить, що програма нестабільна, а звичні дії можуть привести до того, що застосунок завершить роботу і дані, важливі для користувача, цілком ймовірно не збережуться, то використання такого застосунку стає небажаним.

Окрім цього випробування програмного продукту дозволять нам перевірити продукт на наявність дефектів та баг (від англ. «bug»), тобто невідповідність реальної поведінки застосунку наперед визначеним та заданим на етапі проектування вимогам.

4.2.1 Мета випробувань

Метою випробувань є покращення якості застосунку.

4.2.2 Загальні положення

Випробування проводяться на основі наступних документів:

– ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;

4.2.3 Результати випробувань

Наведемо перелік сценаріїв тестування, що необхідні для покриття функціоналу застосунку, у таблицях 4.1 – 4.11. Усі тести були успішно пройдені.

Таблиця 4.1 – Тестовий сценарій «Додавання нових пісень шляхом переміщення нового файлу у директорію музичної бази»

Назва тесту:	Додавання нових пісень шляхом переміщення нового файлу у директорію, що належить до музичної бази
Функція:	Додавання та видалення музичних файлів з директорій, що моніторяться
Кроки:	Очікуваний результат
1) оберіть пунк меню Music->Music Archive Dirs 2) серед списку директорій, що моніторяться, оберіть будь-яку та перейдійть у цю директорію за допомогою застосунку File Explorer 3) додайте у цю директорію будь-який музичний файл, шляхом переміщення його з іншої директорії	Поява сповіщення про нову пісню та кнопки «Fill» для виконання цієї дії на головному вікні застосунку
4) натисніть кнопку «Fill»	Відкривання нового вікна для заповнення інформації про пісню, що була додана

Таблиця 4.2 – Тестовий сценарій «Пошук нової музики»

Назва тесту:	Пошук нової музики
Функція:	Редагування списку мережевих ресурсів , завантаження медіафайлів з мережевих ресурсів

Продовження таблиці 4.2

Кроки:	Очікуваний результат:
1) обрати пунк меню Music->Search New	Повинне відкритися нове вікно «Search New Music».
2) натиснути кнопку «Change resources» 3) зробити «uncheck» для всіх ресурсів та натиснути кнопку «Save»	Кнопка «Start search» повинна бути недоступною.
4) натиснути кнопку «Change resources» 5) обрати щонайменше один ресурс та натиснути кнопку «Save»	Кнопка «Start search» повинна стати доступною.
6) натиснути кнопку «Start search»	Нове вікно, що інформує про початок пошуку нової музики повинне бути відкритим.
7) дочекатися, поки пошук завершиться	Вікно, що інформувало про початок пошуку нової музики повинне закритися. Повинне відкритися нове вікно зі списком знайдених пісень.
8) закрити це вікно	Сповіщення про пісні, що перебувають у «підвішеному» стані повинне появитися на головному вікні. Поруч зі сповіщенням має бути доступна кнопка «Fill».
9) натиснути кнопку «Fill»	Повинне відкритися вікно зі списком знайдених пісень (те ж саме, після виконання кроку 7).

Таблиця 4.3 – Тестовий сценарій «Редагування підписників»

Назва тесту:	Редагування підписників
Функція:	Редагування списку отримувачів
Кроки:	Очікуваний результат:
1) обрати пункт меню People->Subscribers	Відкрите нове вікно «Manage Subscribers»
2) видалити одного підписника зі списку	Список підписників оновлено
3) натиснути кнопку «Save»	Вікно «Manage Subscribers» закрилося.
4) обрати пункт меню People->Subscribers	Список підписників повинен бути зміненим. Попередньо видалений підписник(крок 2) повинен бути відсутнім у списку.

Таблиця 4.4 – Тестовий сценарій «Додавання нових директорій до музичного архіву»

Назва тесту:	Додавання нових директорій до музичного архіву
Функція:	Редагування списку директорій для моніторингу пісень
Кроки:	Очікуваний результат:
1) обрати пунк меню Music-> Music Archive Dirs	Відкрите нове вікно «Music Archive Dirs»
2) в поле вводу ввести невалідний шлях, наприклад: FC123:/fg34/5t 3) натиснути кнопку «Add path»	Видане повідомлення про помилку

Продовження таблиці 4.4

4) ввести валідний шлях до директорії 5) натиснути кнопку «Add path»	Директорія додана до списку директорій, що моніторяться
6) перевідкрити вікно	Список директорій повинен містити новододану директорію
7) закрити вікно, перейти до головного вікна	Сповіщення про пісні, що перебувають у «підвішеному» стані повинне появитися на головному вікні. Поруч зі сповіщенням має бути доступна кнопка «Fill».

Таблиця 4.5 – Тестовий сценарій «Керування шаблонами плейлистів»

Назва тесту:	Керування шаблонами плейлистів
Функція:	Створення/видалення шаблонів плейлистів
Кроки:	Очікуваний результат:
1) обрати пунк меню Playlist->Manage Playlist Patterns	Відкрите нове вікно «Manage Playlist Patterns»
2) натиснути кнопку «Add new»	Відкрите нове вікно «Add New Pattern»
3) заповнити необхідні дані: Name: TEST Tags to include: обрати щонайменше один	Обраний тег зі списку «Tags to include» повинен автоматично бути виключеним зі списку «Tags to exclude»
4) натиснути кнопку «Add»	Діалог «Add New Pattern» закрився, шаблон плейлиста був доданий до списку доступних патернів, що

Продовження таблиці 4.5

	доступний у вікні «Manage Playlist Patterns»
5) видалити будь-який інший доступний паттерн, якщо такий доступний 6) натиснути кнопку «Close» 7) обрати пунк меню Playlist-> Generate New Playlist	Відкрите нове вікно «Generate New Playlist», серед списку доступних патернів доступний новододаний патерн (крок 3-4), патерн, що був видалений у кроці 5, не відображається у цьому списку

Таблиця 4.6 – Тестовий сценарій «Створення нового плейлиста»

Назва тесту:	Створення нового плейлиста
Функція:	Створення плейлистів
Кроки:	Очікуваний результат:
1) обрати пунк меню Playlist-> Generate New Playlist	Відкрите нове вікно «Generate New Playlist»
2) обрати шаблон для генерування плейлиста 3) зробити uncheck для «Use default config» 4) заповнити поля «Duration» та «Songs/Jingle» 5) натиснути кнопку «Generate»	Згенеровано новий плейлист з приблизною тривалістю до заданої. Відкрито нове вікно, яке містить інформацію про список пісень, що входять до цього плейлиста. Також пропонується обрати дату та час, коли згенерований плейлист звучатиме в ефірі.
6) обрати дату та час звучання, натиснути кнопку «OK» 7) перейти до вкладки «Timetable» на головному вікні	Плейлист додано до таймінгу.

Таблиця 4.7 – Тестовий сценарій «Оновлення даних про поточне відтворення»

Назва тесту:	Оновлення даних про поточне відтворення
Функція:	Спостерігання за етапом відтворення на клієнтській стороні
Кроки:	Очікуваний результат:
1) відкрити головний екран застосунку 2) дочекатися моменту, коли на клієнтській стороні почне грати нова пісня	Список поточних пісень оновлюється, підсвічується нова композиція (та, що звучить на клієнтській стороні)

Таблиця 4.8 – Тестовий сценарій «Додавання нової пісні»

Назва тесту:	Додавання нової пісні, автоматично завантаженої з музичного ресурсу, до музичного архіву
Функція:	Заповнення даних про новододані пісні
Кроки:	Очікуваний результат:
1) обрати пунк меню Music->Search New	Повинне відкритися нове вікно «Search New Music».
2) натиснути кнопку «Change resources» та обрати хоча б один ресурс 3) натиснути кнопку «Start search»	Нове вікно, що інформує про початок пошуку нової музики повинне бути відкритим.
4) дочекатися, поки пошук завершиться	Вікно, що інформувало про початок пошуку нової музики закрилося.

Продовження таблиці 4.8

	Відкритися нове вікно зі списком знайдених пісень.
5) серед списку обраних пісень обрати будь-яку	У полях Author таTitle відображається інформація про автора на назву композиції
6) натиснути кнопку «Play»	Композиція відтворюється
7) натиснути кнопку «Stop»	Відтворення композиції припинене
8) натиснути кнопку «Submit»	Відкрився діалог «Fill Song Info»
9) заповнити інформацію про автора та назву пісні, наприклад «ТЕСТ АВТОР», «ТЕСТ НАЗВА» 10) обрати будь-яку директорію з існуючого списку, обрати будь-які теги для копозиції 11) натиснути кнопку «OK»	Вікно закрилося. Композиція відсутня видалена зі списку нових пісень
12) відкрити директорію, що була обрана у кроці 10, за допомогою застосунку File Explorer 13) здійснити пошук композиції ТЕСТ АВТОР - ТЕСТ НАЗВА у цій директорії	Пісня знайдена

Таблиця 4.9 – Тестовий сценарій «Додавання нових підписників»

Назва тесту:	Додавання нових підписників
Функція:	Редагування списку отримувачів
Кроки:	Очікуваний результат:

Продовження таблиці 4.9

People->Subscribers	Subscribers»
2) натиснути кнопку «+»	Відкрите нове вікно «Add New Subscriber»
3) залишити поля вводу пустими, натиснути кнопку «OK»	Повідомлення «Name and Email cannot be empty. Please check provided info.» відображається на діалозі.
4) ввести дані в поле вводу Name, наприклад: Наталія 5) ввести невалідні дані в поле вводу Email, наприклад: n.stupynets@gmail.cooooooom 6) натиснути кнопку «OK»	Повідомлення «Email is not correct. Please check provided info.» відображається на діалозі.
7) ввести валідні дані в поле вводу Email, наприклад: n.stupynets@gmail.com	Вікно закрилося. До списку підписників у вікні «Manage Subscribers» додався новий елемент «Наталія, n.stupynets@gmail.com»

Таблиця 4.10 – Тестовий сценарій «Gmail інформування»

Назва тесту:	Gmail інформування
Функція:	Надсилання повідомлень про зміну бази пісень, перегляд повідомлень про зміну бази пісень
Кроки:	Очікуваний результат:
1) виконати кроки 1-4 з тесту «Додавання нових пісень шляхом	Відкрито діалог «Fill Song Info»

Продовження таблиці 4.10

переміщення нового файлу у директорію, що належить до музичної бази» (таблиця 4.1)	
2) виконати кроки 9-11 з тесту «Додавання нової пісні, автоматично завантаженої з музичного ресурсу, до музичного архіву» (таблиця 4.7) 3) відкрити головне вікно застосунку	Сповіщення про нові пісні, про які не проінформовані підписники, появилося на головному вікні. Поруч зі сповіщенням доступна кнопка «Notify»
4) натиснути кнопку «Notify» 5) здійснити вхід до Gmail-акаунту одного з підписників	Отримано лист з інформацією (назва, автор) про новододану до музичного архіву композицію

Таблиця 4.11 – Тестовий сценарій «Інформація про автора та застосунок»

Назва тесту:	Інформація про автора та застосунок
Функція:	Виведення інформації про автора та застосунок
Кроки:	Очікуваний результат:
1) обрати пунк меню About->About Radio Play	Відкрито нове вікно, в якому доступна інформація про застосунок
2) закрити вікно з інформацією про застосунок 3) обрати пунк меню About->About Author	Відкрито нове вікно з інформацією про автора застосунку

Висновок до розділу

На даному етапі ми організували повноцінну перевірку роботи системи. Тест-кейси, що були розроблені для цієї перевірки не лише допомогли нам здійснити валідацію продукту на даному етапі, а й стануть в нагоді у майбутньому. Наприклад, якщо вже існуючий функціонал буде доповнено новими можливостями, дані тест-кейси будуть використані для того, щоб перевірити чи доповнення функціоналу не вплинуло на поведінку вже існуючого. Також після проведення тестування ми можемо гарантувати, що система стабільна та основний функціонал працює як і вимагалося.

ЗАГАЛЬНІ ВИСНОВКИ

В ході виконання дипломної роботи були розширені теоретичні знання, отримані за період навчання, та застосовані здобуті практичні навички для розв’язання поставленої задачі.

В рамках дипломного проекту була сформульована задача автоматизації робочого місця музичного редактора на радіостанції та встановлені вимоги до її виконання. Також були визначені процеси, що підлягають автоматизації, розроблено спосіб вирішення проблеми, обрані оптимальні технології для реалізації програмного забезпечення в рамках поставленої задачі, розроблена архітектура програмного забезпечення.

Окрім цього, у технологічному розділі було наведено інструкцію користувача та сценарії тестування для перевірки якості продукту.

ПЕРЕЛІК ПОСИЛАНЬ

1. Перелік українських радіостанцій, що використовують програму «Sound Empire» [Електронний ресурс] // Режим доступу: http://www.se2.com.ua/Rus/clients.html.
2. Трегуб В. – Проектування систем автоматизації. Навчальний посібник. 1-ше видання – Київ, «Ліра-К» 2014.
3. Booch G., Rumbaugh J., Jacobson I. – The Unified Modeling Language User Guide. 2nd edition – Boston, «Addison-Wesley Professional» 2005.
4. Qt Documentation for Qt5.12 [Електронний ресурс] // Режим доступу: https://doc.qt.io/qt-5/.
5. Erik T. Ray – Learning XML. 2nd edition – Sebastopol, California, «O'Reilly Media» 2003.
6. Myers G.J. –The Art of Software Testing. 3rd edition – Hoboken, New Jersey, «John Wiley & Sons, Inc» 2011.
7. Gamma E., Helm R., Johnson R., Vlissides J. – Design Patterns: Elements of Reusable Object-Oriented Software. 1st edition – Boston, «Addison-Wesley Professional» 1994.
8. Meyers S. – Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14. 1st edition – Sebastopol, California, «O'Reilly Media» 2014.
9. Straustrup B. – A Tour of C++. 2nd edition – Boston, «Addison-Wesley Professional» 2018.

ДОДАТОК А

Автоматизоване робоче місце музичного редактора

(Найменування програми (документа))

DVD-R

(Вид носія даних)

143 арк, 424 Кб

(Обсяг програми (документа) , арк.) Кб)

Київ – 2019 року

Додаток А

```
//file main.cpp
#include "RadioPlayApplication.h"

int main(int argc, char *argv[])
{
RadioPlayApplication app(argc,argv);
app.init();
return app.exec();
}

// file RadioPlayApplication.h
#ifndef RADIOPLAY_APPLICATION_H
#define RADIOPLAY_APPLICATION_H

#include "network_pack/NetworkManager.h"
#include "email_notifier_pack/EmailNotifier.h"
#include "folder_observer_pack/FolderObserver.h"
#include "songs_pack/SongManager.h"
#include "playlists_pack/PlaylistManager.h"
#include "playlists_pack/PlaylistHistory.h"
#include "ui_components_pack/UIComponent.h"
#include "playlists_pack/PlaylistGenerator.h"
#include "music_archives_pack/MusicArchivesManager.h"

#include <QApplication>

class RadioPlayApplication : virtual public QApplication
{
Q_OBJECT

public:
   RadioPlayApplication(int argc, char *argv[]) : QApplication(argc,argv) {}
   virtual ~RadioPlayApplication(){}

   void init();
        void run();
        void exit();

private:
   void connectComponenents();

   NetworkManager m_networkManager;
   EmailNotifier m_emailNotifier;
   FolderObserver m_folderObserver;
   SongManager m_songManager;
   PlaylistManager m_playlistManager;
   PlaylistHistory m_playlistHistory;
   UIComponent m_uiComponent;
   PlaylistGenerator m_playListGenerator;
```

```
    MusicArchivesManager m_musicArchivesManager;
};
#endif // RADIOPLAY_APPLICATION_H

#include "RadioPlayApplication.h"
void RadioPlayApplication::init()
{
    m_networkManager.init();
    m_emailNotifier.init();
    m_folderObserver.init();
    m_songManager.init();
    m_playlistManager.init();
    m_playlistHistory.init();
    m_uiComponent.init();
    m_musicArchivesManager.init();

    connectComponenents();
}

// UI component
#ifndef UI_COMPONENT_H
#define UI_COMPONENT_H

#include "../general_pack/mystring.h"
#include "ui/mainWindow.h"
#include "ui/dialogsearchnewmusic.h"
#include "ui/dialogwaitfomusicrsearch.h"
#include "ui/dialogmusicresources.h"
#include "ui/musicArciveDirs.h"
#include "ui/dialogmanagesubscribers.h"
#include "ui/RecentlyDownloadedSongsDialog.h"
#include "ui/playlistPatternsDialog.h"
#include "ui/dialogabout.h"
#include "ui/dialoggeneratenewplaylist.h"
#include "ui/dialogmanagemusicarchive.h"
#include "ui/dialoggmailoptions.h"

#include <vector>
#include <map>
#include <QDialog>

class Song;
class UIComponent : public QObject
{
    Q_OBJECT

public:
    UIComponent();
    virtual ~UIComponent();

    void init();
```

```
    void onNewSongAddedToDir(String newFilePath);
    void onCurrentSongChanged(String songId);
    void onCurrentPlaylistChanged(String playlistId);

private:
    void connectUiComponents();

    void openAboutAuthorDialog();
    void openManageResourcesDialog();
    void openManagePlaylistPatternsDialog();
    void openFindNewMusicDialog();
    void openManageMusicArchiveDirsDialog();
    void openGeneratePlaylistDialog();
    void openManageSubscribersDialog();
    void openAboutRadioPlayDialog();
    void openManageCurrentMusicArchiveDialog();
    void openGmailOptionsDialog();

    DialogWaitForMusicSearch       m_waitForSearchDialog;
    RecentlyDownloadedSongsDialog  m_recentlyDownloadedSongsDialog;

    MainWindow                     m_mainWindow;

    ManageSubscribersDialog        m_manageSubscribersDialog;
    MusicArciveDirs                m_musicArchiveDirsDialog;
    DialogSearchNewMusic           m_searchNewMusicDialog;
    PlaylistPatternsDialog         m_playlistPatternsDialog;
    DialogMusicResources           m_musicResourcesDialog;
    DialogAbout                    m_aboutDialog;
    GenerateNewPlaylistDialog      m_generatePlaylistDialog;
    ManageMusicArchiveDialog       m_manageMusicArchiveDialog;
    GmailOptionsDialog             m_gmailOptionsDialog;
};
#endif // UI_COMPONENT_H

// UI component cpp
#include "UIComponent.h"

UIComponent::UIComponent()
    : m_manageSubscribersDialog{&m_mainWindow}
    , m_musicArchiveDirsDialog{&m_mainWindow}
    , m_searchNewMusicDialog{&m_mainWindow}
    , m_playlistPatternsDialog{&m_mainWindow}
    , m_musicResourcesDialog{&m_mainWindow}
    , m_aboutDialog{&m_mainWindow}
    , m_generatePlaylistDialog{&m_mainWindow}
    , m_manageMusicArchiveDialog{&m_mainWindow}
    , m_gmailOptionsDialog{&m_mainWindow}
{
}
```

```
UIComponent::~UIComponent()
{
}

void UIComponent::init()
{
   connectUiComponents();

   m_mainWindow.show();
}

void UIComponent::connectUiComponents()
{
   connect(&m_mainWindow, &MainWindow::openAboutAuthorDialogInititated, this,
&UIComponent::openAboutAuthorDialog);
   connect(&m_mainWindow, &MainWindow::openManageResourcesDialogInitiated,
this, &UIComponent::openManageResourcesDialog);
   connect(&m_mainWindow,
&MainWindow::openManagePlaylistPatternsDialogInitiated, this,
&UIComponent::openManagePlaylistPatternsDialog);
   connect(&m_mainWindow, &MainWindow::openFindNewMusicDialogInitiated, this,
&UIComponent::openFindNewMusicDialog);
   connect(&m_mainWindow,
&MainWindow::openManageMusicArchiveDirsDialogInitiated, this,
&UIComponent::openManageMusicArchiveDirsDialog);
   connect(&m_mainWindow, &MainWindow::openGeneratePlaylistDialogInitiated,
this, &UIComponent::openGeneratePlaylistDialog);
   connect(&m_mainWindow, &MainWindow::openManageSubscribersDialogInitiated,
this, &UIComponent::openManageSubscribersDialog);
   connect(&m_mainWindow, &MainWindow::openAboutRadioPlayDialogInititated,
this, &UIComponent::openAboutRadioPlayDialog);
   connect(&m_mainWindow,
&MainWindow::openManageCurrentMusicArchiveDialogInitiated, this,
&UIComponent::openManageCurrentMusicArchiveDialog);
   connect(&m_mainWindow, &MainWindow::openGmailOptionsDialogInitiated, this,
&UIComponent::openGmailOptionsDialog);
}

void UIComponent::openAboutAuthorDialog()
{
   m_aboutDialog.show();
   m_aboutDialog.showInfoAboutAuthor();
   m_aboutDialog.setFocus();
}

void UIComponent::openAboutRadioPlayDialog()
{
   m_aboutDialog.show();
   m_aboutDialog.showInfoAboutApplication();
   m_aboutDialog.setFocus();
}
```

```
void UIComponent::openManageResourcesDialog()
{
   m_musicResourcesDialog.show();
   m_musicResourcesDialog.setFocus();
}

void UIComponent::openManagePlaylistPatternsDialog()
{
   m_playlistPatternsDialog.show();
   m_playlistPatternsDialog.setFocus();
}

void UIComponent::openFindNewMusicDialog()
{
   m_searchNewMusicDialog.show();
   m_searchNewMusicDialog.setFocus();
}

void UIComponent::openManageMusicArchiveDirsDialog()
{
   m_musicArchiveDirsDialog.show();
   m_musicArchiveDirsDialog.setFocus();
}

void UIComponent::openGeneratePlaylistDialog()
{
   m_generatePlaylistDialog.show();
   m_generatePlaylistDialog.setFocus();
}

void UIComponent::openManageSubscribersDialog()
{
   m_manageSubscribersDialog.show();
   m_manageSubscribersDialog.setFocus();
}

void UIComponent::openManageCurrentMusicArchiveDialog()
{
   m_manageMusicArchiveDialog.show();
   m_manageMusicArchiveDialog.setFocus();
}

void UIComponent::openGmailOptionsDialog()
{
   m_gmailOptionsDialog.show();
   m_gmailOptionsDialog.setFocus();
}
// main window
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
```

```
#include <QMainWindow>
//#include "dialogaboutauthor.h"
//#include "dialogmusicresources.h"
//#include "dialogsearchnewmusic.h"
//#include "dialogwaitfomusicrsearch.h"
//#include "musicArciveDirs.h"
//#include "RecentlyDownloadedSongsDialog.h"
//#include "playlistPatternsDialog.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
   Q_OBJECT

public:
   explicit MainWindow(QWidget *parent = nullptr);
   ~MainWindow();

signals:
   void openAboutAuthorDialogInititated();
   void openManageResourcesDialogInitiated();
   void openManagePlaylistPatternsDialogInitiated();
   void openFindNewMusicDialogInitiated();
   void openManageMusicArchiveDirsDialogInitiated();
   void openGeneratePlaylistDialogInitiated();
   void openAboutRadioPlayDialogInititated();
   void openManageSubscribersDialogInitiated();
   void openManageCurrentMusicArchiveDialogInitiated();
   void openMusicArchiveDirsDialogInitiated();
   void openGmailOptionsDialogInitiated();

private slots:
   void on_switchToTimetableButton_clicked();
   void on_commandLinkButton_clicked();

private:
   Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H

#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QPalette>

MainWindow::MainWindow(QWidget *parent) :
   QMainWindow(parent),
```

```
        ui(new Ui::MainWindow)
    {
        ui->setupUi(this);
        setWindowIcon(QIcon{":/images/images/mainImage.png"});
        ui->pictureLabel->setPixmap(QPixmap{":/images/images/mainImage.png"});

        connect(ui->actionAbout_author, &QAction::triggered, this,
&MainWindow::openAboutAuthorDialogInititated);
        connect(ui->actionManage_music_resources, &QAction::triggered, this,
&MainWindow::openManageResourcesDialogInitiated);
        connect(ui->actionManage_playlist_patterns, &QAction::triggered, this,
&MainWindow::openManagePlaylistPatternsDialogInitiated);
        connect(ui->actionFind_new, &QAction::triggered, this,
&MainWindow::openFindNewMusicDialogInitiated);
        connect(ui->actionMusic_archive_dirs, &QAction::triggered, this,
&MainWindow::openManageMusicArchiveDirsDialogInitiated);
        connect(ui->actionGenerate_playlist, &QAction::triggered, this,
&MainWindow::openGeneratePlaylistDialogInitiated);
        connect(ui->actionManage_subscribers, &QAction::triggered, this,
&MainWindow::openManageSubscribersDialogInitiated);
        connect(ui->actionAbout_RadioPlay, &QAction::triggered, this,
&MainWindow::openAboutRadioPlayDialogInititated);
        connect(ui->actionManage_current_music_archive, &QAction::triggered, this,
&MainWindow::openManageCurrentMusicArchiveDialogInitiated);
        connect(ui->actionMusic_archive_dirs, &QAction::triggered, this,
&MainWindow::openMusicArchiveDirsDialogInitiated);
        connect(ui->actionGmail_options, &QAction::triggered, this,
&MainWindow::openGmailOptionsDialogInitiated);
    }

    MainWindow::~MainWindow()
    {
        delete ui;
    }

    void MainWindow::on_switchToTimetableButton_clicked()
    {
        ui->stackedWidget->setCurrentIndex(1);
    }

    void MainWindow::on_commandLinkButton_clicked()
    {
        ui->stackedWidget->setCurrentIndex(0);
    }

    // file EmailNotifier.h
    #ifndef EMAIL_NOTIFIER_H
    #define EMAIL_NOTIFIER_H

    #include "../xml_parsers_pack/XMLParserFactory.h"
    #include "../general_pack/mystring.h"
```

```
#include "windowsIncludes.h"
#include <windows.h>
#include <wininet.h>
#include <set>
#include <map>

#pragma comment( lib, "wininet" )

class EmailNotifier final
{
public:
   EmailNotifier() = default;
   ~EmailNotifier() = default;

   void init();
   bool notifyAll();
   void addReceiver(String receiver, String name);
   void removeReceiver(String receiver);
   void setEmailText(String text);

private:
        bool hasInternetConnection()
        {
                return InternetCheckConnection("http://www.google.com",
FLAG_ICC_FORCE_CONNECTION, 0);
        }

   std::map <String,String> m_mails; //receivers' name and email map
   LCString m_myEmail
   LCString m_myPassword;
   String m_emailText;

};
#endif // EMAIL_NOTIFIER_H

#include "EmailNotifier.h"

bool EmailNotifier::notifyAll()
{
        CkMailMan mailman;

                bool success = mailman.UnlockComponent("30-day trial");
                if (success != true) {
                        return false;
                }

                //  Set the SMTP server.
                mailman.put_SmtpHost("smtp.gmail.com");

                mailman.put_SmtpUsername(m_myEmail.data());
```

```
		mailman.put_SmtpPassword(m_myPassword.data());

		mailman.put_SmtpSsl(true);
		mailman.put_SmtpPort(465);

	for (auto &mail : m_mails)
	{
		//  Create a new email object
		CkEmail email;

		email.put_Subject("RadioPlay data received");
		email.put_Body(mailText.data());

		auto fromWho = String("RadioPlay <") + m_myEmail + String(">");
		email.put_From(m_myEmail.data());
		success = email.AddTo("RadioPlay", mail.data());

		success = mailman.SendEmail(email);
		if (success != true) {
			return false;
		}

		success = mailman.CloseSmtpConnection();
		if (success != true) {
			return false;
		}
		return true;
	}
}

void EmailNotifier::addReceiver(String receiver, String name)
{
	m_mails.insert(receivesr, name);
}

void EmailNotifier::removeReceiver(String receiver)
{
   m_mails.emplace(receiver);
}

void EmailNotifier::setEmailText(String text)
{
	m_emailText = text;
}

// mystring.h
#ifndef MYSTRING_H
#define MYSTRING_H

#include <string>
```

```
#ifdef _UNICODE
typedef std::wstring String;
#else
typedef std::string String;
#endif
#endif

// file IMusicArchive.h
#ifndef I_MUSIC_ARCHIVE
#define I_MUSIC_ARCHIVE

#include "../network_pack/NetworkManager.h"
#include "../songs_pack/Song.h"
#include <vector>

class IMusicArchive
{
public:
   IMusicArchive();
   virtual ~IMusicArchive();

   virtual bool search(std::vector<Song>& newSongs) = 0;
};
#endif // I_MUSIC_ARCHIVE

//parsers files
#ifndef XML_PARSER_FACTORY_H
#define XML_PARSER_FACTORY_H

#include "ParserType.h"
#include "../general_pack/mystring.h"

class XMLParserBase;

class XMLParserFactory final
{
public:
        XMLParserFactory();
        virtual ~XMLParserFactory();

   XMLParserBase* createParser(ParserType parserType, String pathToXML);
};
#endif //XML_PARSER_FACTORY_H

#ifndef XML_PARSER_BASE_H
#define XML_PARSER_BASE_H

#include "../general_pack/mystring.h"

class XMLParserBase
{
```

```
public:
   XMLParserBase(String path);
   virtual ~XMLParserBase();
   virtual bool parseXML();
private:
   String m_xmlFilePath;
};
#endif //XML_PARSER_BASE_H

#ifndef SONG_MANAGER_XML_PARSER
#define SONG_MANAGER_XML_PARSER
#include "XMLParserBase.h"
class SongManagerXMLParser : public XMLParserBase
{

public:
   explicit SongManagerXMLParser(String pathToXML);
   virtual ~SongManagerXMLParser();
         bool parseXML();
};
#endif // SONG_MANAGER_XML_PARSER

#ifndef PLAYLIST_MANAGER_XML_PARSER_H
#define PLAYLIST_MANAGER_XML_PARSER_H

#include "XMLParserBase.h"

class PlaylistManagerXMLParser : public XMLParserBase
{
public:
   PlaylistManagerXMLParser(String pathToXML);
   virtual ~PlaylistManagerXMLParser();
         bool parseXML();
};
#endif // PLAYLIST_MANAGER_XML_PARSER_H

#ifndef PLAYLIST_HISTORY_XML_PARSER
#define PLAYLIST_HISTORY_XML_PARSER

#include "XMLParserBase.h"

class PlaylistHistoryXMLParser : public XMLParserBase
{

public:
   PlaylistHistoryXMLParser(String pathToXML);
   virtual ~PlaylistHistoryXMLParser();
         bool parseXML();
};
#endif // PLAYLIST_HISTORY_XML_PARSER

#ifndef MONITORED_FOLDERS_XML_PARSER_H
#define MONITORED_FOLDERS_XML_PARSER_H
#include "XMLParserBase.h"
class MonitoredFoldersXMLParser : public XMLParserBase
```

```
{
public:
   MonitoredFoldersXMLParser(String pathToXML);
         virtual ~MonitoredFoldersXMLParser();
         bool parseXML();
};
#endif //MONITORED_FOLDERS_XML_PARSER_H

#ifndef EMAIL_INFO_XML_PARSER
#define EMAIL_INFO_XML_PARSER
#include "XMLParserBase.h"

class EmailInfoXMLParser : public XMLParserBase
{
public:
   EmailInfoXMLParser(String pathToXML);
         virtual ~EmailInfoXMLParser();
         bool parseXML();
};
#endif // EMAIL_INFO_XML_PARSER

#ifndef DIALOGABOUTAUTHOR_H
#define DIALOGABOUTAUTHOR_H

#include "ui_dialogabout.h"
#include <QDialog>

class DialogAbout : public QDialog
{
   Q_OBJECT

public:
   explicit DialogAbout(QWidget *parent = nullptr);
   ~DialogAbout();

   void showInfoAboutAuthor() { ui->stackedAboutWidget->setCurrentIndex(0); }
   void showInfoAboutApplication() { ui->stackedAboutWidget->setCurrentIndex(1); }

private slots:

   void on_pushButton_clicked();

private:
   Ui::DialogAbout *ui;

};

#endif // DIALOGABOUT_H

#include "dialogabout.h"

DialogAbout::DialogAbout(QWidget *parent) :
   QDialog(parent),
   ui(new Ui::DialogAbout)
{
```

```
    ui->setupUi(this);
    setWindowIcon(QIcon(":/images/images/mainImage.png"));
    QPixmap authorPhotoPixmap(":/images/images/myPic.jpg");
    ui->authorPhotoLabel->setPixmap(authorPhotoPixmap);
}

DialogAbout::~DialogAbout()
{
    delete ui;
}


void DialogAbout::on_pushButton_clicked()
{
    this->close();
}

#ifndef DIALOGMANAGESUBSCRIBERS_H
#define DIALOGMANAGESUBSCRIBERS_H

#include <QDialog>
#include <QTableWidgetItem>

namespace Ui {
class ManageSubscribersDialog;
}

class ManageSubscribersDialog : public QDialog
{
    Q_OBJECT

public:
    explicit ManageSubscribersDialog(QWidget *parent = nullptr);
    ~ManageSubscribersDialog();

signals:
    void subscribersChanged(QList<QTableWidgetItem*>);

private slots:
    void on_deleteElementButton_clicked();
    void on_addNewButton_clicked();

private:
    Ui::ManageSubscribersDialog *ui;
};

#endif // DIALOGMANAGESUBSCRIBERS_H

#include "dialogmanagesubscribers.h"
#include "ui_dialogmanagesubscribers.h"
#include "ui_dialogaddnewsubscriber.h"

#include <QDialog>

ManageSubscribersDialog::ManageSubscribersDialog(QWidget *parent) :
```

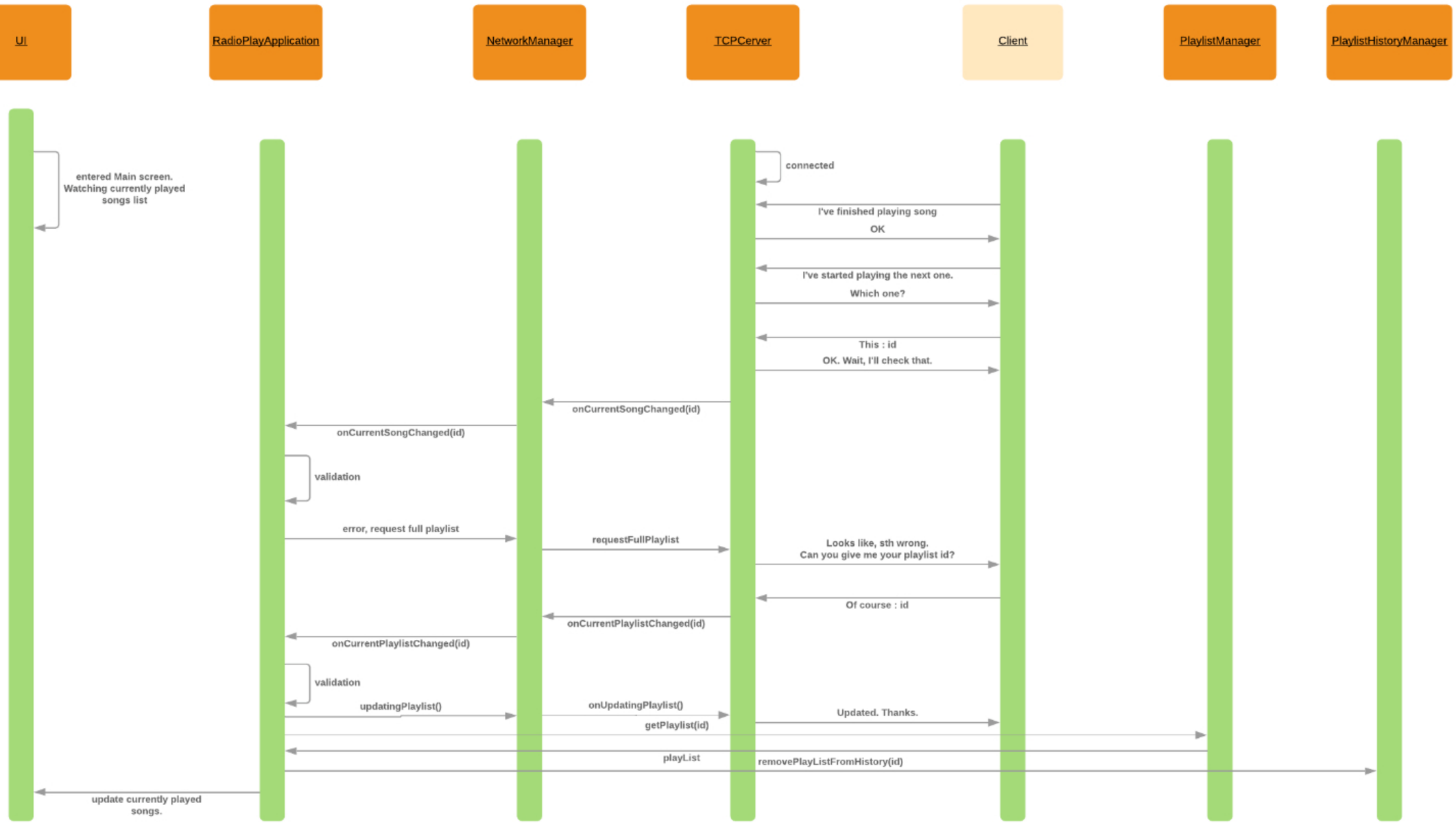
```
    QDialog(parent),
    ui(new Ui::ManageSubscribersDialog)
{
    ui->setupUi(this);
}

ManageSubscribersDialog::~ManageSubscribersDialog()
{
    delete ui;
}

void ManageSubscribersDialog::on_deleteElementButton_clicked()
{
    if (ui->subscribersTableWidget->selectedItems().size() != 0)
        ui->subscribersTableWidget->removeRow(ui->subscribersTableWidget->currentRow());
}

void ManageSubscribersDialog::on_addNewButton_clicked()
{
Ui::AddNewSubscriberDialog addNewUi;
QDialog *addNewDialog = new QDialog(this);
addNewUi.setupUi(addNewDialog);
addNewUi.errorsWidget->setHidden(true);
addNewDialog->show();
}

#ifndef DIALOGMUSICRESOURCES_H
#define DIALOGMUSICRESOURCES_H

#include <QDialog>
//#include "../RadioPlayBack/musicresourcesmanager.h"

namespace Ui {
class DialogMusicResources;
}

class DialogMusicResources : public QDialog
{
    Q_OBJECT

public:
    explicit DialogMusicResources(QWidget *parent = nullptr);
    ~DialogMusicResources();
private:
    unsigned int getResourcesData();
    void writeResourcesData();
private slots:
    void on_cancelButton_clicked();
    void on_saveButton_clicked();
signals:
    void resourcesDataChanged();
private:
    Ui::DialogMusicResources *ui;
    unsigned int m_resourcesData;
};
```

```
#endif // DIALOGMUSICRESOURCES_H

#include "dialogmusicresources.h"
#include "ui_dialogmusicresources.h"
#include <QFile>
#include <QTextStream>
#include <QDir>
#include "../../general_pack/mystring.h"

DialogMusicResources::DialogMusicResources(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::DialogMusicResources)
{
    ui->setupUi(this);
    setWindowIcon(QIcon(":/images/images/mainImage.png"));

    m_resourcesData = getResourcesData();
    if (m_resourcesData)// & IncludeResources::FreeMusicArchive)
        ui->freeMusArchiveCheckBox->setChecked(true);
    else
        ui->freeMusArchiveCheckBox->setChecked(false);
}

DialogMusicResources::~DialogMusicResources()
{
    delete ui;
}

void DialogMusicResources::on_cancelButton_clicked()
{
    this->close();
}

void DialogMusicResources::on_saveButton_clicked()
{
    if (ui->freeMusArchiveCheckBox->isChecked())
        m_resourcesData = ++m_resourcesData;//|= IncludeResources::FreeMusicArchive;
    else
        m_resourcesData = ++m_resourcesData;//&= ~IncludeResources::FreeMusicArchive;

    writeResourcesData();
    emit resourcesDataChanged();

    this->close();
}

unsigned int DialogMusicResources::getResourcesData()
{
    QString data;
    QString fileName(QDir::currentPath() + QString("/data/musicResources.txt"));
    String val = fileName.toStdWString();

    QFile file(fileName);
    if(file.open(QIODevice::ReadOnly))
```

```
    {
        data = file.readAll();
    }
    file.close();

    if (data.isEmpty())
        return 0;

    return data.toUInt();
}

void DialogMusicResources::writeResourcesData()
{
    QString output = QString::number(m_resourcesData);
    QFile file(QDir::currentPath() + QString("/data/musicResources.txt"));
    QString filepath = QDir::currentPath() + QString("/data/musicResources.txt");
    if(!file.open(QIODevice::ReadWrite))
    {
        return;
    }

    QTextStream stream( &file );
    stream << output << endl;
    file.close();
}


// music networsk resouces\
#ifndef MUSICRESOURCESMANAGER_H
#define MUSICRESOURCESMANAGER_H
#include "../FreeMusicArchive.h"
#include <QObject>

enum IncludeResources
{
    FreeMusicArchive = 1<<0
};

class MusicArchivesManager:public QObject
{
Q_OBJECT

public slots:
    void searchNewMusic();

private:
    MusicArchivesManager();
    ~MusicArchivesManager();

private:
    unsigned int getIncludedResourcesData();

private:
    FreeMusicArchive *freeMusicArchive;
};
```

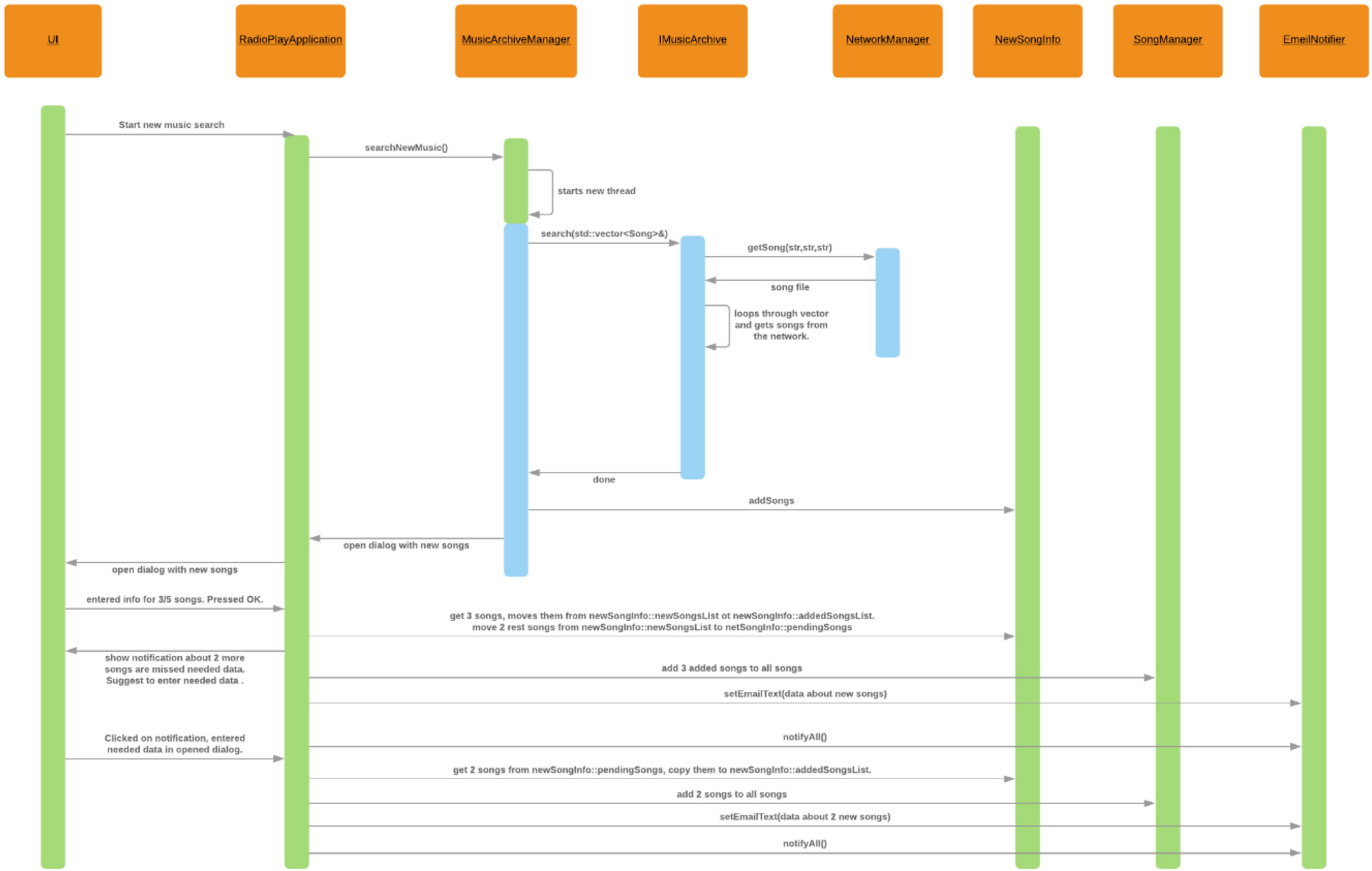
```
#endif // MUSICRESOURCESMANAGER_H
// cpp  file
MusicArchivesManager::MusicArchivesManager()
{
   freeMusicArchive = new FreeMusicArchive();
}

MusicArchivesManager::~MusicArchivesManager()
{
   delete freeMusicArchive;
}

unsigned int MusicArchivesManager::getIncludedResourcesData()
{
   QString data;
   QString fileName(QDir::currentPath() + QString("/data/musicResources.txt"));

   QFile file(fileName);
   if(file.open(QIODevice::ReadOnly))
   {
      data = file.readAll();
   }
   file.close();

   if (data.isEmpty())
      return 0;

   return data.toUInt();
}

void MusicArchivesManager::searchNewMusic()
{
   auto includedResources = getIncludedResourcesData();
   auto resources = std::vector <MusicResource*>{};

   // if resource was checked, then we search new music there
   // future resources must be added here
   if (includedResources & FreeMusicArchive)
      resources.push_back(freeMusicArchive);

   for (auto resource : resources)
   {
      resource->search();
    }
}
```

ДП ІС-з4119.1722-с.ССП



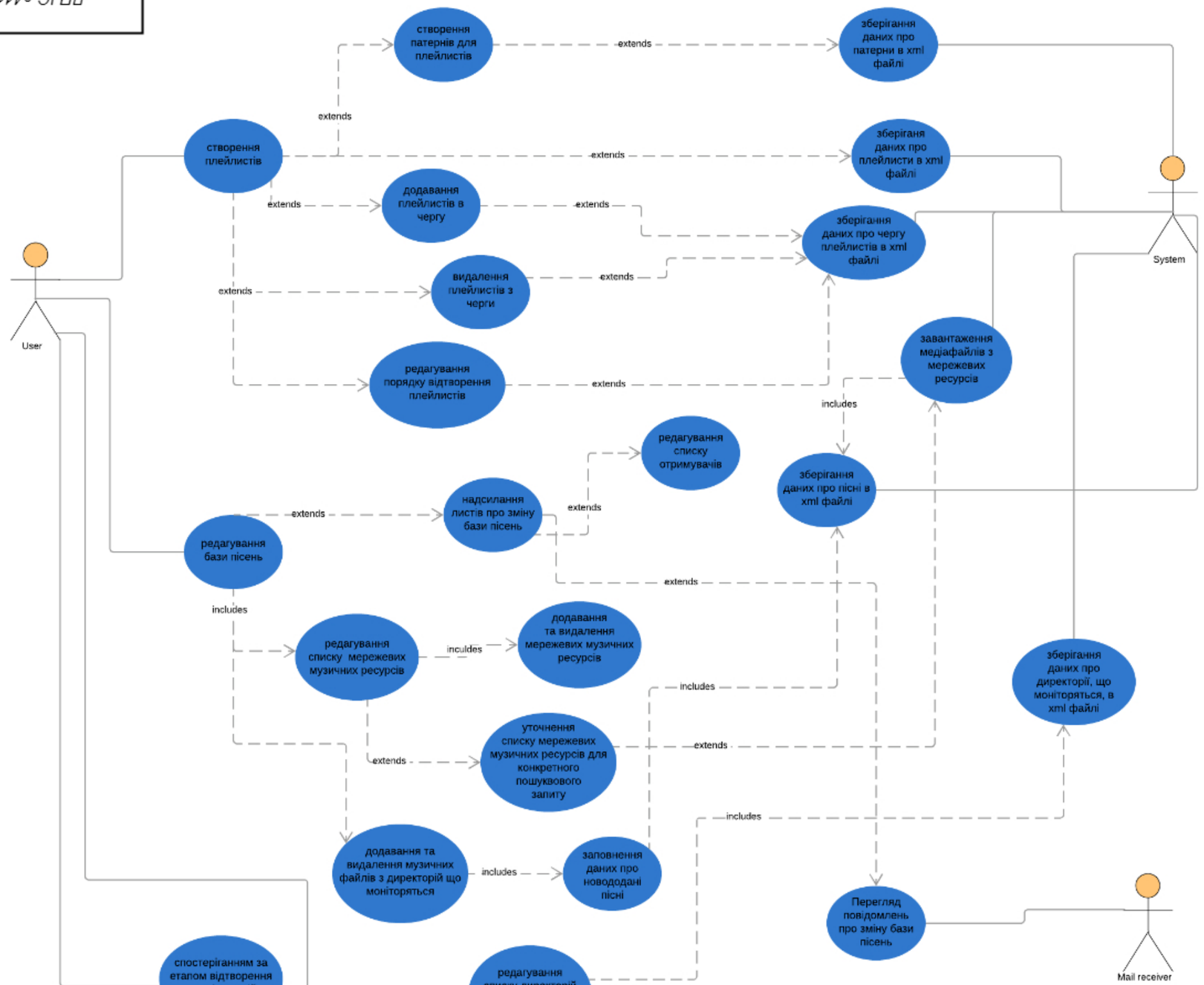
					ДП ІС-з4119.1722-с.ССП		
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна послідовності оновлення користувацького інтерфейсу внаслідок комунікації між клієнтом та сервером	Літера	Маса / Масштаб
Розробив		Ступинець Н.В.					
Перевірив		Халус О.А.				Аркуш 1	Аркушів 1
Т. кон.							
					Автоматизоване робоче місце музичного редактора	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-341	
Н. кон.		Телишева Т.О.					
Затвердив		Павлов О.А.					

ДП ІС-34119.1722-с.ССП



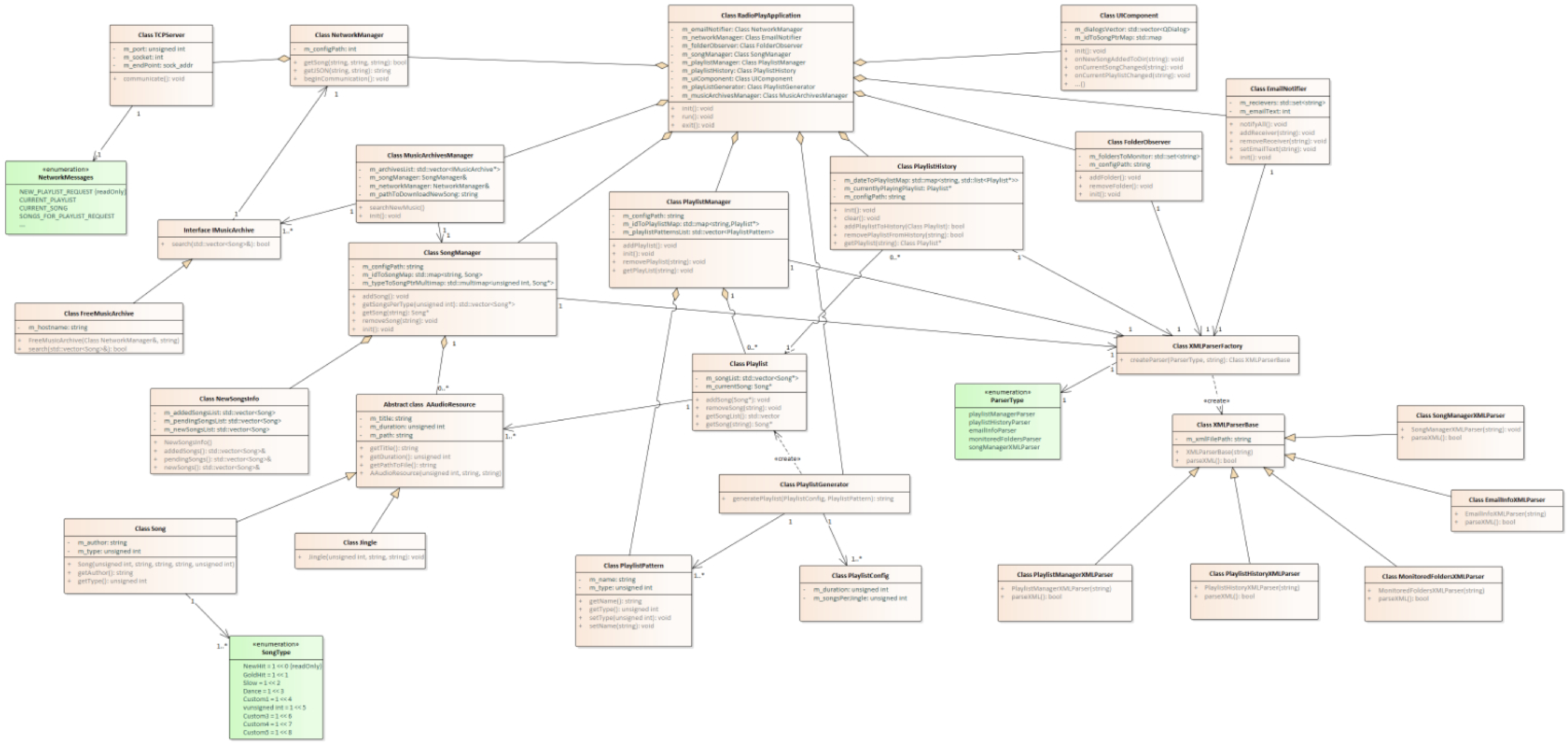
					ДП ІС-34119.1722-с.ССП	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна послідовності пошуку нових пісень у мережі	Літера / Маса / Масштаб
Розробив		Ступинець Н.В.				
Перевірив		Халус О.А.				Аркуш 1 / Аркушів 1
Т. кон.						
Н. кон.		Телишева Т.О.			Автоматизоване робоче місце музичного редактора	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-341
Затвердив		Павлов О.А.				

ДП ІС-з4119.1722-с.ССВ



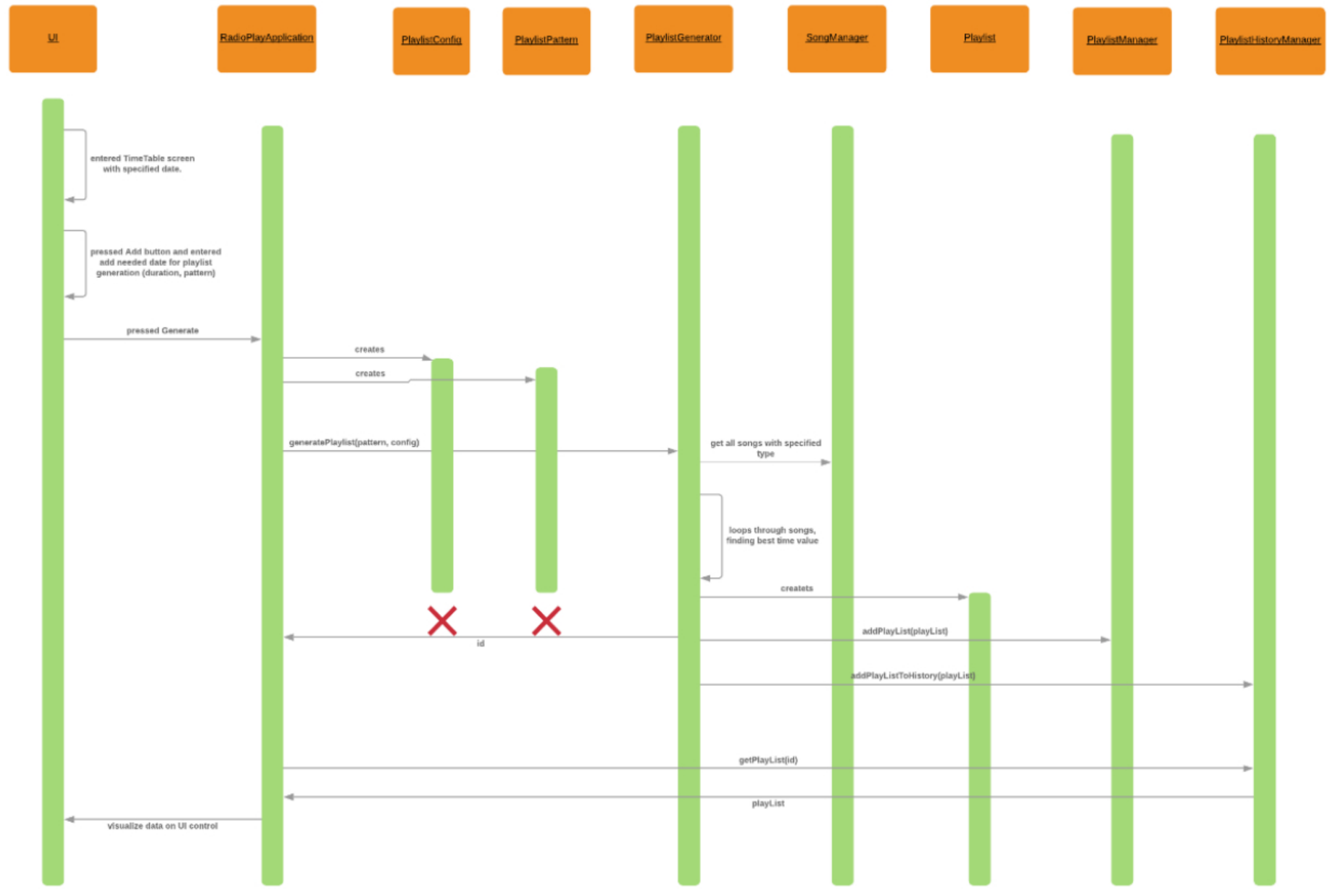
					ДП ІС-з4119.1722-с.ССВ	
					Схема структурна варіантів використання	Літера: — Маса: — Масштаб: —
Зм.	Арк.	№ документа	Підпис	Дата		Аркуш 1 / Аркушів 1
Розробив		Ступинець Н.В.				
Перевірив		Халус О.А.				
Т. кон.						
Н. кон.		Телишева Т.О.			Автоматизоване робоче місце музичного редактора	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-341
Затвердив		Павлов О.А.				

ДП ІС-з4119.1722-с.ССК



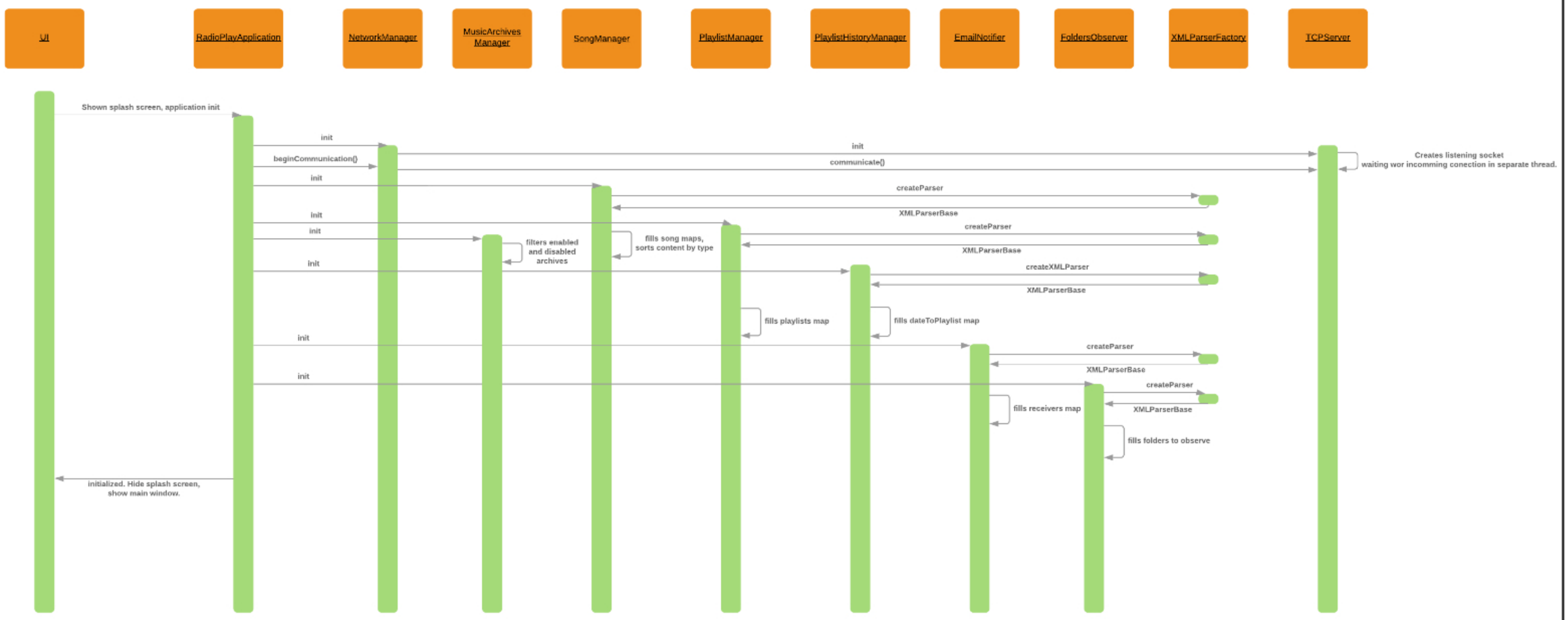
					ДП ІС-з4119.1722-с.ССК			
					Схема структурна класів програмного забезпечення	Літера	Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата				
Розробив		Ступинець Н.В.						
Перевірив		Халус О.А.						
Т. кон.						Аркуш 1	Аркушів 1	
					Автоматизоване робоче місце музичного редактора	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-341		
Н. кон.		Телишева Т.О.						
Затвердив		Павлов О.А.						

ДП ІС-з4119.1722-с.ССП



					ДП ІС-з4119.1722-с.ССП			
					Схема структурна послідовності додавання згенерованого плейлиста до розкладу відтворення плейлистів	Літера	Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата				
Розробив		Ступинець Н.В.				Аркуш 1	Аркушів 1	
Перевірив		Халус О.А.						
Т. кон.					Автоматизоване робоче місце музичного редактора	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-341		
Н. кон.		Телишева Т.О.						
Затвердив		Павлов О.А.						

ДП ІС-з4119.1722-с.ССП



					ДП ІС-з4119.1722-с.ССП			
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна послідовності ініціалізації класів при запуску застосунку	Літера	Маса	Масштаб
Розробив		Ступинець Н.В.						
Перевірив		Халус О.А.				Аркуш 1	Аркушів 1	
Т. кон.								
Н. кон.		Телишева Т.О.			Автоматизоване робоче місце музичного редактора	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-з41		
Затвердив		Павлов О.А.						